(2)

Adapting Software Development Policies to Modern Technology

unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS n/a
2a. SECURITY CLASSIFICATION AUTHORITY USAF, AFSC/CA	3. DISTRIBUTION/AVAILABILITY OF REPORT Dist. A. Approved for public release; unlimited distribution.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE ----------	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) n/a	5. MONITORING ORGANIZATION REPORT NUMBER(S) n/a
6a. NAME OF PERFORMING ORGANIZATION AIR FORCE STUDIES BOARD NATIONAL RESEARCH COUNCIL 6b. OFFICE SYMBOL (If applicable) n/a	7a. NAME OF MONITORING ORGANIZATION HQ AFSC/CA
6c. ADDRESS (City, State, and ZIP Code) 2101 Constitution Avenue, N.W. Washington, D.C. 20418	7b. ADDRESS (City, State, and ZIP Code) Andrews AFB, MD 20334
8a. NAME OF FUNDING/SPONSORING ORGANIZATION HQ AFSC 8b. OFFICE SYMBOL (If applicable) n/a	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Andrews AFB, MD 20334	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. n/a; PROJECT NO. n/a; TASK NO. n/a; WORK UNIT ACCESSION NO. n/a

11. TITLE (Include Security Classification)

ADAPTING SOFTWARE DEVELOPMENT POLICIES TO MODERN TECHNOLOGY (U)

12. PERSONAL AUTHOR(S) Committee on Adapting Software Development Policies ...

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
final	FROM 6/88 TO 8/88	1989 July	88

16. SUPPLEMENTARY NOTATION

the

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	software; software reliability; SEE; software acquisition
12	08		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The problem of developing reliable software that is capable of performing its intended function has persisted since advent of digital computers. Subject has been reviewed many times. Are newer methods of software development now being introduced for large, high-technology systems outstripping conventional software acquisition techniques and policies?

The committee reviewed recent software studies to learn why they did not adequately solve problem. Also, committee assessed current and past development programs, investigated methods for the user and developer to work more closely, evaluated software process models as alternatives to the waterfall model, evaluated incremental development, evolutionary development, prototyping, etc.

Software acquisition (KR) Keywords: software reliability (KR)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Vernon H. Miles, Sr., Director AFSB	202/334-3531	n/a

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

(2)

Adapting Software Development Policies to Modern Technology

Committee on Adapting Software Development Policies
to Modern Technology
Air Force Studies Board
Commission on Engineering and Technical Systems
National Research Council

NATIONAL ACADEMY PRESS
Washington, D.C. 1989

NOTICE: The project that is the subject of this report was approved by the Governing Board of the National Research Council, whose members are drawn from the councils of the National Academy of Sciences, the National Academy of Engineering, and the Institute of Medicine. The members of the committee responsible for the report were chosen for their special competences and with regard for appropriate balance.

This report has been reviewed by a group other than the authors according to procedures approved by a Report Review Committee consisting of members of the National Academy of Sciences, the National Academy of Engineering, and the Institute of Medicine.

The National Academy of Sciences is a private, nonprofit, self-perpetuating society of distinguished scholars engaged in scientific and engineering research, dedicated to the furtherance of science and technology and to their use for the general welfare. Upon the authority of the charter granted to it by the Congress in 1863, the Academy has a mandate that requires it to advise the federal government on scientific and technical matters. Dr. Frank Press is president of the National Academy of Sciences.

The National Academy of Engineering was established in 1964, under the charter of the National Academy of Sciences, as a parallel organization of outstanding engineers. It is autonomous in its administration and in the selection of its members, sharing with the National Academy of Sciences the responsibility for advising the federal government. The National Academy of Engineering also sponsors engineering programs aimed at meeting national needs, encourages education and research, and recognizes the superior achievements of engineers. Dr. Robert M. White is president of the National Academy of Engineering.

The Institute of Medicine was established in 1970 by the National Academy of Sciences to secure the services of eminent members of appropriate professions in the examination of policy matters pertaining to the health of the public. The Institute acts under the responsibility given to the National Academy of Sciences by its congressional charter to be an adviser to the federal government and, upon its own initiative, to identify issues of medical care, research, and education. Dr. Samuel O. Thier is president of the Institute of Medicine.

The National Research Council was organized by the National Academy of Sciences in 1916 to associate the broad community of science and technology with the Academy's purposes of furthering knowledge and advising the federal government. Functioning in accordance with general policies determined by the Academy, the Council has become the principal operating agency of both the National Academy of Sciences and the National Academy of Engineering in providing services to the government, the public, and the scientific and engineering communities. The Council is administered jointly by both Academies and the Institute of Medicine. Dr. Frank Press and Dr. Robert M. White are chairman and vice chairman, respectively, of the National Research Council.

This report represents work under Contract No. F49620-87-C-0122 between the United States Air Force and the National Academy of Sciences.

Copies of this report are available from:

Air Force Studies Board
National Research Council
2101 Constitution Avenue, N.W.
Washington, D.C. 20418
(202) 334-3531

and from the Defense Technical Information Center, Alexandria, Virginia.

AIR FORCE STUDIES BOARD STAFF

Vernon H. Miles, Sr., Director
Donald L. Whittaker, Editor/Staff Associate
Terrie Noble, Administrative Associate
Katherine H. Atkins, Secretary

STATEMENT OF TASK

The problem of developing reliable software that is capable of performing its intended function has persisted since the advent of the digital computer. The subject has been reviewed many times by various groups, including the Air Force Studies Board (AFSB), yet the Commander of the Air Force Systems Command states that it is his primary problem in systems development. Four years ago the AFSB studied software reliability. The Board wants to know how effective their previous study was and why it and other recent studies on the subject have not resolved the problems with software development. The answer may be that newer methods of software development now being introduced for large, high-technology systems are outstripping conventional software acquisition techniques and policies. These newer methods are directed at coping with the realization that software cannot be defined and controlled under conventional precepts of baseline management (as outlined in MIL-STDs 483, 490, 999, 1521, 1679, and DoD-STD 2167). Three areas where the system may break down are:

1. Software architects need access to experimental testing of systems alternatives before they are in a position to write sharply stated requirements or make optional design selections.

2. User interfaces are poorly handled or ignored under the present procurement system.

3. Design inadequacies or outright failures in other system elements are corrected by software changes. These occur late in the procurement cycle and are invariably handled in an ad hoc manner outside the normal procurement actions.

Solutions are emerging to handle these difficulties but they cannot be introduced into the procurement system because they all violate the basic principle of an early, formal definition of design. This study may provide new concepts for maintaining control of acquisitions during the early phases that do permit software architects to work effectively.

To better understand the problems in software development and help derive solutions, the committee will:

- review recent software studies and determine why they have not resolved the software acquisition problem,

- assess some current and past development programs and identify software deficiencies,

- investigate methods for the software user and software developer to work more closely together in co-evolution of the task,

- consider the role of prototyping and evolution in the software development process,

- evaluate software process models as alternatives to the waterfall model (e.g., the spiral model of software development, or software first) especially as regards their appropriateness to high-tech procurements, and

- determine how adequate controls (e.g., formal derivation history) for testability, reliability, and survivability can be made part of an incremental development.

Note: This statement of task proved a very serviceable basis for the committee's study. However, we must point out that studies can seldom if ever *resolve* problems of these kinds, only *suggest* means of resolution. The aforementioned design inadequacies and failures must be handled by normal procurement actions, for there is no magic alternative. We recognize that there are always ad hoc deviations from the preliminary plans for many complex systems.

The last of the six tasks listed above appears to presuppose a mechanism to be used during incremental development, and is addressed explicitly in Section 3.3.3.

CONTENTS

EXECUTIVE SUMMARY

THE PROBLEM

Today computer software controls the operation of most weapon systems; ten years ago it did not. Accordingly, software-related development has steadily increased in importance to Air Force Systems Command (AFSC). As an issue, it has arisen more and more frequently in recent times, as evidenced by at least 17 "software studies" initiated by the Air Force, Department of Defense (DoD), and U.S. General Accounting Office (GAO) over the past 15 years. The Committee on Adapting Software Development Policies to Modern Technology was asked to review these earlier studies and find out why, in spite of their efforts, they did not solve software development problems. From this review, we reached two conclusions: (1) the importance and complexity of software have grown faster than could be adequately handled by the prevailing management and technical methods, and (2) while this growth might have been predicted, its consequences for personnel and programs were not fully anticipated.

In addition, the committee was asked to examine programs past and present to determine their software deficiencies. We accomplished this review with the cooperation of representatives from the AFSC product division, Air Force Logistics Command (AFLC), and the using commands. In doing so, we found few problems for which software alone could be blamed.

Most problems that surfaced as software shortcomings were due in fact to not having well-developed requirements or to impractical acquisition constraints. Successful acquisition of software-intensive systems was due primarily to early reduction of risks and to software-knowledgeable management. Many key system "players" in both government and industry are weak in software knowledge. Also, software experts are frequently not expert in systems. Software has not been properly recognized as an item that takes a relatively long time to develop; instead development of appropriate software has often been initiated late in the system life cycle. Although software has been used to overcome some hardware limitations (but with time or cost overruns), some hardware and system limitations are beyond software's ability to compensate.

Overall, we addressed three main areas: software development *processes*, the *people* needed to manage and carry out these processes, and the *technology* that underlies process evolution. Each of these areas demands more attention now that software is making a major transition, joining integrated circuits at the center of information technology.

ADA, STARS, AND SEI

Ada, the Department of Defense computing language, is a combination of "good news" and "bad news." It is truly valuable for constructing large software programs; however, its use for a system demands a proven compiler, which is more complex than those for older, simpler languages. For real-time applications it is comparable to most other high-order programming languages; however, its performance with existing hardware is less than that possible with the use of machine language program-

ming. Waivers from the Ada standard will make economic and operational sense in some cases; but in each case cost and performance issues must be adequately understood.

Three different efforts involving the Defense Advanced Research Projects Agency's STARS (software technology for available, reliable systems) program support this committee's recommendations, particularly in relation to software engineering environments (SEEs). STARS, however, should not be expected to produce near-term results; from an Air Force perspective, STARS results may appear in system proposals within a few years, but not before then.

The Software Engineering Institute (SEI) is a federal contract research center, and should be viewed as an additional rather than a primary resource. SEI staff should complement those employed directly by the Air Force. AFSC should work with SEI to take advantage of SEI's capabilities in contractor assessment and SEEs.

FINDINGS

Five software issues occupied the committee's attention: (1) the (system) acquisition process, (2) acquisition practices, (3) software engineering, (4) in-house software expertise, and (5) software technology.

The Acquisition Process. The process model for software development defined by DoD-STD-2167A is the so-called "waterfall" model. This model, often held to be the *only* acceptable model, assumes *non-iterative* development between steps. The waterfall model is an open loop model, often characterized by unrealistic specifications, and measured by documentation rather than demonstrable results. Contractors slavishly follow this flawed model, which only adds to the problem.

The waterfall model is satisfactory *only* for precedented systems, i.e., those for which substantial implementation experience exists. In other cases, it is highly risky and does not emphasize safeguards in the definition of requirements, evolution of system design, and development or maintenance of effective design teams. *In short, alternative models are needed for the development of unprecedented systems.*

We recommend appropriate application of alternative acquisition processes. Such alternative processes should include a demonstration/validation (Dem/Val) phase, a competitive first-phase procurement, or both. Also, a warm, and therefore responsive, industrial base should be maintained by funding application-specific software technology programs.

Acquisition Practices. In examining acquisition practices, we found that DoD-STD-2167 and MIL-STD-1521 (the waterfall model) and other specifications and standards often were imposed with little concern for the characteristics of the particular acquisition. Firm fixed price contracting is still being imposed on development of some *unprecedented* systems.

Although many program offices are calling for more user involvement, it appears that a more tailored user involvement -- the right number of the right people at the right time -- is required. Tailoring of acquisitions should be universally and intelligently applied, yet there can be more risk than reward in such tailoring; without the experience and knowledge base to guide them, program offices become uncomfortable and ignore tailoring or leave it to the contractors. Lack of timely attention to software development often puts software *advanced development* in the midst of full-scale system development.

We recommend that AFSC issue a policy statement and implement workshops in tailored acquisition, devoting special emphasis to software. The handbooks supporting Standards 2167A and 2168 should be published expeditiously as part of this initiative, which should also address user and maintainer involvement. Current Air Force regulations suggest using the software maintainer (e.g., Air Force Logistics Command [AFLC]) for Independent Validation and Verification (IV&V); this was reported by AFLC to have worked well and merits wider consideration. Also, AFSC should avoid firm fixed price contracting in the acquisition of unprecedented systems, unless adequate prior risk reduction efforts are made.

Software Engineering. Software engineering requires disciplined application of a proven set of tools and techniques (collectively referred to when computer-based as a *software engineering environment*, or an SEE) during software development and later, after delivery with the system. Automated tools are needed to control information throughout the system life cycle, first by the contractor and then by the maintainer. Certain SEE tools can also help AFSC program managers understand and monitor progress, collect data, and plan programs. Developers' software engineering, which is discussed in AFSC pamphlet 800-43 (31 January 86), may consist principally of the developmental tool set used in software development. Such a tool set will include a compiler, debugger, library, test support, etc. However, for unprecedented systems, SEEs must include a design exploration tool set including one or more very high level languages (VHLL) and simulators for testing. For management purposes, yet another tool set may be employed, containing process models, scheduling and tracking software, and acquisition management communications.

A few varieties of SEE are available today, but too few to meet the projected needs of NASA and the Department of Defense. NASA is investing a large budget in an SEE to be developed for use by contractors of its Space Station program; DoD's Joint Avionics Integration Working Group plans to provide an SEE, as government-furnished equipment, to contractors of constituent programs. Large, geographically distributed system and software developments demand this kind of computer resource, which can also provide tools to help AFSC's software acquisition activities.

We believe that AFSC should first, require contractors to use an appropriate SEE for all but the smallest software-development acquisitions. Second, developers of unprecedented systems should be required to prove their possession of and knowledge concerning suitable design exploration tools. Third, AFSC should acquire those elements of an SEE that will support computer-based documentation, and build up a management tool set. Fourth, contractors should be required to deliver software engineering data in machine-readable form that can be manipulated by AFSC's document preparation tools.

In-house Software Expertise. In examining the personnel resources AFSC can bring to bear on software issues, we identified only about 320 software-specialty officers at work in the Command. (The number of software-knowledgeable civilians or of systems engineers with strong software skills could not be determined but is not believed large enough to meet present and future needs.) Today few senior people are in this specialty field, though some software-knowledgeable officers are employed in more visible specialties promising better chances for promotion. The need for expertise is clearly on the rise and already exceeds the number of experienced personnel available, and

there is no Air Force or AFSC program in operation that can meet that need soon enough.

We recommend that experts now in the command be identified and that some of them be organized into one or more software systems engineering "advisory teams" based on knowledge and experience. These individuals should be used at key stages before, during, and after contract awards involving significant software components. Further efforts should be made to identify and develop system engineers with strong software skills and software engineers with system competence. Such individuals are particularly important in AFSC, although they are needed also in industry. These qualified individuals are a scarce resource; they should be intensively managed through appropriate career progression and special monitoring. Military software specialists should expect a reasonable promotion path, say to grade 0-6 after 20 years in service, with opportunity for general officer promotions.

Software Technology. We agree with earlier studies that software acquisition problems are largely managerial; however, they do have important technical aspects. For example, many commercially available software tools and research results are slow to be used in the Air Force acquisition process. Compared to technologies believed to be important to the Air Force's technological "edge," software technology is underfunded by a factor of 2 to 4; this contrast should be addressed. We recommend using SEEs as primary vehicles for technology transfer of important software process and management technologies. We also recommend continuing to fund programming language evolution for VHLLs and application-oriented languages appropriate to AFSC products.

We urge the Air Force to establish a production-funded program similar to MANTECH but for software production technology pertinent to particular systems or system classes. Software process technologies should be commercialized since that would improve available tools at less cost to the government. We suggest that the software technology base be expanded at the fastest rate it can sensibly proceed, over a period of five years or more, to bring it into closer parity with other critical technology areas.

1.0 INTRODUCTION

1.1 BACKGROUND

Today computer software controls the operation of most weapons systems; ten years ago it did not. Developing reliable software that is able to perform its intended function has been a problem since the advent of the digital computer. The Commander of the Air Force Systems Command states that it is now the most serious obstacle to systems development. At the Commander's request, the Air Force Studies Board established the committee on Adapting Software Development Policies to Modern Technology in May 1988 to help the Air Force devise new precepts of software baseline management so that software architects can design a more effective and reliable product. The work of the committee began in early June and ended in late August 1988.

1.2 STUDY APPROACH

To better understand the problems in software development and find solutions, we:

- reviewed recent software studies to determine why they have not solved the software acquisition problem,
- assessed some current and past development programs to identify software deficiencies,
- investigated ways that the software user and developer can work more closely together in co-evolution of the task,
- considered the role of prototyping and evolution in the software development process,
- evaluated alternative software development process models, and
- determined how adequate controls for testability, reliability, and survivability can be made part of an incremental development.

The committee followed a twofold approach. First, we identified at least 17 earlier software studies initiated by the Air Force, Department of Defense, or General Accounting Office during the last 15 years, including one 1984 Air Force Studies Board study of software reliability. We reviewed these studies to determine why they did not lead to solution of software development problems (see Appendix A, Previous Studies with Brief Summary of Recommendations).

Second, we visited various Air Force bases to hear briefings from software development managers about the deficiencies of current and past Air Force programs. In June, we visited the Electronic Systems Division at Hanscom Air Force Base to hear presentations on software in command and control systems, and visited the Aeronautical Systems Division at Wright-Patterson Air Force Base to hear presentations on software in aircraft systems. In July, members of the committee met at Warner Robins Air Logistics Center to learn some of the views of the personnel who maintain electronic warfare systems. During a two-week summer session in Irvine, California, the committee heard presentations by Space Systems Division, Strategic Air Command, Tactical Air Command, Air Force Space Command, Air Force Communications Command, and STARS. The committee then wrote this report. (See Appendix B for the Schedule of Committee Meetings.)

Overall, the committee addressed three main areas: software development *processes*, the *people* needed to manage and carry out these processes, and the *technology* that underlies process evolution. Each area needs more attention now that software is making its major transition to join integrated circuits as a central technology for information and control of systems. This study provides new concepts for maintaining control of acquisitions during the early phases so that software architects can work effectively.

2.0 BACKGROUND

2.1 PERSPECTIVES

2.1.1 Introduction

The central subject of this report is the acquisition of software in Air Force systems. Software is hardly a novel topic in Air Force-sponsored studies. A short history of related past studies and their major recommendations follows in Section 2.2.1. If computer hardware and software technology were stagnant fields, there would be no need for a new study. But if one field changes, the environment of the other is changed. Hardware has evolved at a very rapid pace that has been unimpeded by predictions that software may be reaching its limits in advancement. Software technology, as we shall see, has also evolved in several important directions. Perhaps most important, the objectives that software will be expected to meet have recently increased dramatically. The Air Force Systems Command's traditional roles are changing, and its Air Force clients (Air Force Logistics Command and the using commands) are changing also. Amid such a pattern of major change there has been no difficulty in finding new challenges.

2.1.2 Software is Ubiquitous

Although application software has been a requisite of Air Force computer utilization since its early involvement with commercial computers, only very recently have software controlled aircraft (e.g., the later blocks of F-15s and F-16s) been added to Air Force inventories. The 8-12-year or longer acquisition cycles that have become characteristic of the Department of Defense (DoD) in peacetime has tended to mask the fact that most systems now contemplated or under development are predominantly software-controlled.

This situation is not accidental. Software control affords degrees of flexibility unknown in hardware-defined systems. Processes for software development, modification, and testing must be made to operate reliably and affordably under timely and responsive configuration control. When these processes are mastered, the Air Force will possess unprecedented abilities to respond to threat changes or uncertainties and to incorporate new technologies (software or hardware) with speed and confidence.

2.1.3 Software Research and Development Investment

Until the 1960s, software was a small part of system research and development (R&D) cost, reflecting the small amount of software in use. Many digital systems of that period were "hard-wired" and required no software. Today, however, for all but the simplest digital functions in electronics, software control is preferred. This approach reflects not only the ability to alter details or parameters late in development or during field operation, but also the far greater power of a stored-program controlled processor over hard-wired logic.

2.1.4 Software = System Personality

In earlier systems, the "personality" of the system was fixed by hardware design decisions; in modern systems its main principles are defined by software. The many modes of a modern airborne attack radar, for example, are defined

solely by software. The F-16 system's ground-attack and air-attack modes are selected in its central computer and define operation of both digital and analog subsystems.

With control of the system operation has come increased dependence on software. The most important *system* decisions often map directly into software decisions. Individuals responsible for a system's software must participate in the system-level design considerations. Software has become the principal instrument for implementing the design, from the top level down to small details.

2.1.5 Integration of Software-Intensive Systems

Traditionally, complex Air Force systems are integrated from many subsystems. Attempts to reduce numbers of subsystems as a means of reducing system complexity can be only partially successful, for none of the Air Force's industrial suppliers is knowledgeable in all aspects of flight control, propulsion and fuel management, navigation, target detection, weapon management, communications, electronic warfare, and the like. Also, there is essential analog hardware (seekers, engines, control surface actuators, etc.) whose integration into digital systems is understood only by specialists. The net result is that integration of the hardware (and software) products of several or many firms will still be required in major systems.

Integration within a software-dominated system, for comparable functional complexities, is no more difficult and in some ways simpler than in a hardware-dominated system. Certain traditional principles still apply: make subsystem interfaces as "clean" (logically specific, unambiguous, uniform) as is practical, and align hardware and software interfaces (that is to say, programs defining a subsystem in detail should operate within that subsystem).

Given intelligent integration of subsystems that are designed to operate reliably in a closely integrated system, integration of software-controlled systems ultimately may be simpler than integration of older hardware-intensive systems because most subsystem testing can be done by simulation of the rest of the system. We are, however, far from that state of affairs, in part because software developers (and others) lack knowledge of the physics associated with the system operation. Incomplete interface definition and other shortcomings further complicate the simulation.

2.1.6 Important Changes in Computer and Digital Systems Arenas

The trend toward software controlled systems is occurring in an industrial environment that features rapid increases in computer power and available memory capacity, doubling capacity every three years, and reducing cost so that the newer and better systems cost little more than their predecessors.

Although high-level programming languages (HLLs) have existed for more than three decades, the small processing power and memory capacity of most of the computers embedded in weapon systems discouraged use of HLLs. As a result, to this date most of the weapon system software maintained by the Air Force Logistics Command (AFLC) is still written in assembly (machine) languages. Air Force determination in the early 1970s to require that the JOVIAL language be used to program all systems was premature. HLLs broke into Air Force front-rank weaponry with the modern J-3B version of JOVIAL in the B1 and F-16. More recently, Ada was selected by DoD as the HLL of choice. Despite a slow start on Ada by the Air Force, the Air Force and its contractors

have made the needed ideological changes, and most perceive benefits through the use of Ada.

After decades of computer industry focus principally on computer hardware evolution, there has recently been a significant growth of emphasis on software. This change has come about, in part, because of the relatively mature nature of conventional processor design, and the realization that highly paralleled or concurrently executing processor assemblages require software advances if they are to become more widely useful.

Exploratory activities in artificial intelligence (AI) represent one cutting edge of software engineering. AI-based activities are underway that will make important contributions to Air Force computer application. The anthropomorphic character attributed to AI ("machines that think like people") may have unintentionally misinformed some laymen as to the reasonable expectations for next-generation computers. The user-oriented thrust of AI was, however, appropriate for addressing human-computer interfaces, an area where progress previously had been quite slow.

2.1.7 Software-Related Technology

Software is unlike other major components of Air Force systems, because it requires design and testing but no manufacturing, and because "maintenance" means repair of design errors and software enhancements.

Software itself has no wear-out modes. It reveals few parallels to features of hardware that might be used through analogy to improve it. The hardware in which it is used, however, determines the ultimate performance software can attain, although performance can be degraded by poor software design.

Software for numerical analysis has been around so long that it draws little research attention. This cannot be said for symbolic (object, text, data base) analysis, because the human-machine interface needs substantial improvement to permit it to deal with semantically complex problems. This area defines a major area of software research, some under the banner of AI.

The software technology of most direct importance to this committee's objectives is that related to the development, testing, integration, and modification of software -- in essence, the manufacturing and testing tools for software programs. Although some of these are now traditional (e.g., compilers, editors, debuggers, operating systems), others are not. There is critical need for tools that allow system and software developers to describe system operation in languages that are unambiguous, complete, readable by humans and computers alike, and that can express concepts of a computer application in ways meaningful to application-area experts.

2.1.8 Related Changes in System Design

System designers must respond to the trend toward software-defined systems. The most important aspect of the trend, from their point of view, is the additional system integration that is possible and will become desirable to users.

For example, in most military aircraft, real-time integration is a crew task. Sensor data is presented in the cockpit, and the crew manipulates controls that actuate engine, airframe, and other mechanisms. The introduction of many new electronic subsystems since World War II has greatly increased crew workloads while providing greater fire power and survivability. But crews have been reduced while crew tasks have in-

creased. Some subsystems are internally integrated so that the crew has only to turn them on. Nothing approaching maximum subsystem interdependence--data from each sensor being used by many other subsystems, for example--has been approached in practice. This interdependence is expected in the next generation of aircraft.

Integration must rely heavily on software, since hardware lacks the needed flexibilty to respond readily to essential change. Integration must be under the control of the overall system designers. Integration also affects hardware design since most integration needs represent additional computer workloads and memory capacity inside or outside the subsystems. Not only performance, but also testing will become software-intensive. Unless systems engineers are adequately conversant with the capabilities and limitations of software, this increased integration will *not* be successful. Likewise, software developers must appreciate and contribute to system design decisions.

2.1.9 Software Considerations in the Department of Defense Versus the Commercial Software Market

The DoD is probably the largest U.S. user of custom software. Aside from commercial computer programs used mainly in administrative activities or by personal computers, most of the software acquired by DoD is unique to particular DoD needs. DoD is also a large, though not dominant, customer for commercial software such as operating systems or data base management programs. The U.S. work force of computer programmers is much more involved in commercial data processing and personal computing applications than in DoD programs. In commercial data processing, the COBOL language remains popular, and C remains popular in personal computing, while Ada seems unlikely to take over either of those areas.

Real-time programs characteristic of weapon system software are high unit-cost items. Because of time-urgency of national security operations, stringent real-time limitations will continue to be placed on Defense software despite the increasing processing power. This appears certain to keep alive machine (assembly) language requirements, even where most processing can be done under less severe timing constraints using Ada-based software.

The issue of providing *operating system programs* for DoD mission-critical computer resources has long been avoided or ignored. The purpose of an operating system is to coordinate and organize the assignment of computer system resources such as memory, storage, and input/output. Commercial operating systems for large mainframes lack the real-time qualities often demanded by DoD, and there have been no operating systems available commercially for most of the many small militarized computers DoD has installed during the past two decades. Just as important, hardware resources managed by DoD weapon system computers lack the interface commonality expected in the design of commercial operating system programs. Thus, few computers assigned to time-critical DoD applications have enjoyed the design flexibility possible with a full-capability operating system.

While the flexibility of operating systems can (if proper care is lacking) lead to indeterminacies of performance unacceptable in time-critical Defense applications, more and more Defense systems will benefit from the utility of the features usually associated with operating systems. Some briefers showed mild interest in militarization of AT&T's popular, portable UNIX operating system. The designers of Ada anticipated that operating-system-like

features would be built into an "Ada environment" that (like an operating system) would require hardware-unique parts. Although this software feature did not emerge as critical during our study, it could become so as all Air Force systems grow in complexity during the next few years.

2.1.10 Software in the Air Force

Probably the most important consideration for the management of software development in the Air Force is its wide variety of applications.

On the one hand, Electronic Systems Division (ESD) and Space Systems Division (SSD) manage a large number of software developments based on commercially available hardware and operating systems operating in surface or underground locations. Aeronautical Systems Division (ASD), Munitions Systems Division (MSD), and SSD manage many system developments involving both hardware and real-time software. Aside from systems developed through AFSC, computer installations serve administrative functions in all commands. For software developed within the Air Force, AFSC is near the bottom of a list that includes at least AFLC, Tactical Air Command, Strategic Air Command, and Air Force Communications Command.

Air Force commands and units view systems and system functions from several different perspectives (as user, operator, developer, maintainer, and/or buyer). AFSC is assigned the role of buyer for a large share of these systems. Our assessment of AFSC's software problems was complicated by different perceptions of need by different Air Force commands and units. Not surprisingly, some of these views conflict. The design considerations for a national strategic warning system are obviously far different from those for an avionic radar warning system, although both are critically important.

Accordingly, AFSC management should not always expect its product divisions to see software problems in the same light. It also follows that *no single procurement method or acquisition strategy can possibly serve for such a variety of situations.* Thus, while in this report we propose promotion of incremental development and of prototyping, neither should be recommended as a universal practice.

2.1.11 User Considerations

AFSC plays the role of smart buyer in software and systems acquisition. Traditionally, AFSC has included representatives of the users (for example, present or former fighter pilots) in its acquisition cadres or evaluation groups to provide needed user input.

Today's more complex software is often expected to embody interests of its users more broadly, including facilities for rapid modification of operational flight programs. Such modification is only one area in which operating commands and AFLC maintainers have strong interest. The inherent flexibility of software is made even more manifest by HLL programming; user commands plan to take advantage of the vaunted software flexibility, and have the personnel to articulate those desires if not the in-place resources to carry them out.

Even with extensive control mechanisms, efforts to rapidly modify software without extensive testing usually produce errors, and often disasters. Control mechanisms and testing alone do not constitute an acceptable user-reprogramming capability, especially where changes may be made independently in several combat theaters. More robust approaches to providing mission- and theater-dependent functions and

parameters will need to be evolved.

2.2 PREVIOUS STUDIES

2.2.1 Discussion

Appendix A is a summary of 17 documents that were available and reviewed for the study. Some were written by study groups outside the Air Force; others were sponsored by the Air Force Studies Board and the Air Force Scientific Advisory Board. Each of the latter addressed software matters at least twice directly; in the case of the AFSB, both studies were 3-week summer studies. For each item, a brief summary or paraphrase of the recommendations is given.

The 17 reports are not a complete data base of everything written on software in the last decade, nor do they include many private studies for the Air Force that addressed software, nor do they reflect continuing interactions of computer-informed people with Air Force managers and officers. They do reflect a compendium of what study groups (some of which were specifically chartered by the Air Force) believed the problems and solutions to be. Importantly, the studies involved people who knew the Air Force as an organization and institution, and could hopefully bridge the gap between the insider environment and the outside world.

Table A-1 in Appendix A summarizes the recommendations of all studies that addressed the software issue directly, and catalogs the recommendations by subject. A few topics were not addressed by more than one study, but these topics were either of specific concern at the time or were directed items for the study group. These topics include STARS, the DoD software initiative, and the DoD programming language Ada.

Other topics have developed only in recent years, or have become significant only in recent years. Thus, they have been addressed only by corresponding groups; included are (1) integrity, the definition of which has currently become important in the context of computer security for information systems and is now just being threshed out; (2) reusable software, which has begun to make technical sense and show realistic feasibility just in the last few years; (3) post-deployment lifecycle support, a subject with which the Air Force has had to deal for most of a decade but which has now grown to such magnitude that it is a major driver for software concerns; and finally (4) tools and environments for software development.

The tools-and-environment issue was addressed by the 1977 Heffron study but lay quiescent for six years until the Vick group singled it out for major attention. The long interval probably reflects the growth of the technical foundation for the subject in the Ada community and other groups. The early identification of the issue probably reflects the influence of the AT&T Bell Telephone Laboratories in the 1977 activity. At the time, Bell Labs was developing large and complex software systems.

Excluding the ones just noted, the six stalwarts that have been considered by nearly every study are (1) definition of requirements, (2) acquisition (procedures), (3) the development process, (4) personnel (quality and quantity), (5) policy (acquisition and management), and (6) management (at both project and oversight levels). While it is dangerous to work from an incomplete data base of documents, observations and conjectures about each can illuminate the progress and the causes of the software problem.

2.2.2 Definition of Requirements

First noted by the Heffron study in 1977, the problem of defining requirements lay dormant for six years until the Air Force Scientific Advisory Board revived it in 1983. While the problem undoubtedly was recognized throughout the interval, the lack of activity probably reflected a gestation period in which ideas for trying to improve the up-front requirements process were developed. It is also possible that the Air Force needed the time to really understand software requirements, a much more detailed, tedious, and potentially trouble-making phase than for hardware projects.

2.2.3 Acquisition

Also noted by the Heffron study, acquisition had intermittent attention by other groups. Both DoD and Air Force policies were changing through the mid-1970s and mid-1980s, system (and therefore, software) responsibility was fragmented in the Air Force. The Air Force Comptroller held responsibility for non-weapon system (i.e., non-embedded) acquisitions, but there was also some diffusion of responsibility throughout the major commands (MAJCOMs); the Deputy Chief of Staff for Research and Development had responsibility for embedded systems. While personnel specialty codes (AFSCs) for computeroriented people existed, career progression was not perceived as satisfactory. Also, Congress was trying to sort out and legislate the acquisition of embedded and non-embedded systems.

In retrospect, the mid-1970s to mid-1980s seem to be a period of trying and shakeout. The Air Force has established a focal point for computer and for command, control, communications, and intelligence (C^3I) at the Assistant Chief of Staff level. But this organization does not have primary responsibility for acquisition of development systems; that remains in the R&D side of the house. The R&D side, working with AFSC and the other Joint Logistics Commanders (AFLC and Army and Navy counterparts), drafted and redrafted standards, the latest wrinkle being the concept of tailoring, that is, deleting documentation requirements from DoD-STD 2167A rather than modifying them. Tailoring more broadly can include changes of program phasing as well.

2.2.4 The Development Process

The development process was addressed early in the 1978 Walden study under Air Force Scientific Advisory Board auspices, but it languished for about five years. That committee apparently concluded that the Air Force had not used contemporary software development methods already in use in the private sector; or perhaps the Air Force could not compel its contractors to use such methods. The methodology that the study groups had in mind tended to center on software-oriented research efforts but also on the commercial/industrial segment where quite different motivations influence decisions and managers.

2.2.5 Personnel

Personnel is a chronic problem in and out of the Air Force. This was noted by the 1977 Heffron study, and then not again until the early 1980s. In the interim, the Air Force had been trying to improve its internal management of scarce resources, but the issues identified in the most recent Brooks study have persisted through the years.

The overwhelming growth of computer-related activity throughout society has kept the pressure on the Air Force to retain its people. At the same time the increasingly pervasive computer lit-

eracy and skill that younger personnel bring with them, plus the proliferation of personal computers, have acted to make otherwise "computer outsiders" into informed and experienced practitioners.

The best efforts of the Air Force to achieve an adequate supply of computer software and systems people have kept the Air Force from falling hopelessly behind. In its own jargon, the Air Force has barely managed to stay "on the power curve."

2.2.6 Policy

Of all the issues, policy has received the most persistent attention over the years, probably because any study group will believe implicitly that there must be something wrong with the rules of the game or the problems would not exist. The many recommendations reflect experimentation on the part of the studies to try out innovative ideas.

2.2.7 Management

Curiously, the management issue was not mentioned in the earlier studies although the personnel issue was. Evidently, oversight management and monitoring was not considered as critical as technical project management (which probably reflected the emphasis on personnel). Due to the increasing complexity of weapon systems and their software and also because of the magnitude of the projects, the importance of management control on budget, schedule, and performance surfaced as a demand for tools and procedures to keep the management hierarchy informed of progress and alert for incipient problems.

2.2.8 Summary

One might be tempted to argue that any reasonably informed computer person would, if told that software was in trouble, identify the six issues just summarized. On the other hand, the diversity of the groups that produced the relevant reports, the variability of the individuals involved, the time span over which the studies were conducted, and the changing context in which succeeding studies were done all collectively argue that the six do indeed constitute a valid, although not necessarily complete, characterization of the Air Force software dilemma.

On the other hand, it would not be fair to conclude that the Air Force has failed to respond to the recommendations. While its response has appeared to some observers to be inadequate-(mostly because the software problems still exist) many things have been done. Communications and computer people have been consolidated into a common organization. Communications and computers officers have been consolidated into a common set. A focal point for software has been created at the Assistant Chief of Staff level.

Either the response of the Air Force either has not been vigorous or sustained enough, or the problem has escalated faster than the Air Force has been able to react. At this point the Air Force, and AFSC in particular, has two choices: take additional actions either on past remedial actions or on new ones, or learn how to live with and plan around the problem.

In judging the last decade of studies and recommendations and the Air Force response to them, one must remember how vigorously the software aspects of systems have advanced in complexity and scope; and one always must keep in mind how demanding the Air Force has become for weapon and system capability that can be provided only through complex software.

2.3 ADA, STARS, AND THE SOFTWARE ENGINEERING INSTITUTE

Ada, Software Technology for Available, Reliable Systems (STARS), and the Software Engineering Institute (SEI) are not topics of this study, but because they are clearly related to this subject we were asked to provide our views on them.

2.3.1 Ada

Ada should be used in applications that have traditionally employed procedure-oriented languages such as JOVIAL, FORTRAN, and COBOL, or more machine-oriented languages. Fielded applications that have successfully and productively used problem-oriented languages such as Atlas for automated test equipment (ATE), LISP, and Prolog for AI and knowledge-based applications, should not be forced to move to Ada at this time.

Even where Ada is the appropriate language, the following should be considered before Ada is used on specific programs:

- The availability of one or more production quality compilers that meets the performance requirements of the program. The program office and/or the prime contractor should have adequate benchmarks to determine whether:
  - Time and table limitations are compiled that can support the size of the system, the size of the software teams, the design and implementation strategies, and any other aspect critical to the success of the development effort. We recommend the stated limits be verified with the benchmarks.
  - The real-time performance (timing, memory, bus bandwidth margins, etc.) is within the proper bounds. Benchmarks specific to the application are needed.
  - The relevant items in Chapters 13 and 14 of MIL-STD-1815 (the DoD Ada specifications) are properly and efficiently implemented. These chapters concern real-time directives to the compiler, input/output (I/O), foreign (non-Ada) software, etc. These features are not now fully tested by the Ada Compiler Validation Capability and are highly implementation dependent.
- The availability of quick reaction compiler support to provide fixes or temporary "workarounds." Compilers will always have errors, but if corrections must wait until the next "release" (usually handled at 6- to 12-month intervals), an immature compiler is not usable and even a seasoned one may be questionable in certain applications.
- The adequacy of the development team's skill and knowledge of Ada and the compiler(s) to be used. Since Ada usually presents a severe "culture shock" to uninitiated development teams, learning and using Ada for the first time is too big a risk to take for a critical system development. Contractors should demonstrate the proficiency of the specific team(s) to be used before award.

Further, the design approach (e.g., use of special language features such as tasking, exception handling, and generics) should be clearly established and followed during the design. Also, designers and programmers must have an understanding of the real-time and

memory implications of their source code. The use of benchmarks or instructors from the compiler vendor can help serve this understanding. Quality assurance teams, armed with automated tools, can help maintain consistency with the desired approach, as well as identify possibly wasteful use of resources (such as timing and memory).

- The experience and competence in Ada of the System Program Office personnel in Ada. This knowledge is necessary to insure that the above issues are properly handled and the risks are adequately attended to by the contractor.

2.3.2 The STARS Program

The STARS program, DoD's corporate software technology program, has recently sharpened its goals and implemented them in terms of three parallel parts:

- "competing primes,"
- (forty-funded) software technology studies, and
- "shadow" programs.

"Competing primes" is the centerpiece of the STARS program. Three prime system contractors have been funded, in parallel and competitively, to build Ada-oriented SEEs particular to three specific application areas that include advanced tool sets supporting the software development cycle with particular emphasis on integrated design support. Each contractor is to maximize the use of subcontractors, with the aim of bringing the leading software technology groups into this competition, thus widening the program's influence out into the software industry. The strategy also emphasizes the use and development of commercial software tools with the intent of influencing the commercial software arena as well as that of aerospace software.

The forty-funded software technology studies began in 1987. Current plans call for them to be folded into the competing primes activity. Each study involves advanced computer science technologies that are all ultimately focused on needed Ada-oriented applications, many of which are Air Force applications.

"Shadow" programs are a daring initiative aimed at simultaneously overcoming resistance to Ada and developing productivity metrics to quantify the benefits of Ada. The concept is to fund modest parallel developments in ongoing development programs planned before the advent of Ada, to introduce Ada skills and benefits.

AFSC procurements should eventually benefit to some degree from the well-meshed STARS program combination. Benefits should be broadly based, with a wide range of software engineering and computer science initiatives of interest to AFSC. All are Ada-oriented, a correct and useful orientation for the times. STARS is a program aimed at significant long-term payoffs; its results should become noticeable in the early 1990s.

2.3.3 Software Engineering Institute

The Software Engineering Institute (SEI) is a recently established FFRDC (federally funded R&D center) that, along with MITRE and Aerospace Corporation, can be used to improve AFSC acquisitions. SEI's principal role is to improve and hasten transition of software technology from its creative beginnings to direct use in acquisitions leading to lowered costs, quicker schedules, and reduced latent errors. Like the other DoD activities, SEI is pledged to the proliferation and success of Ada.

Later in this report we recommend that AFSC (1) compel their software suppliers to use software prototypes to demonstrate architectural design worthiness, (2) enforce use of SEEs, (3) develop its in-house personnel resources, (4) aggressively tailor acquisition regulations, (5) bring the user into the procurement earlier, and (6) improve management of software by installing and using an appropriate process model for each acquisition. In one way or another, SEI is working on all of these recommendations. Its work on software process models is the leading research effort in this important area. Its tools for the evaluation of potential software contractors (developed jointly with MITRE) are also outstanding and will likely play a major role in upgrading acquisition practices. The SEI is a new institution that is well positioned to help AFSC improve its acquisition practices. AFSC must, however, establish channels into SEI and its DoD taskmasters that will allow AFSC concerns to be understood and SEI's results to move easily into Air Force procurements.

3.0 STRATEGIES FOR SOLUTIONS

3.1 OVERVIEW

The difficulties of acquiring software systems can be eased by addressing three major elements: the *processes*, the *products*, and the *people*. The processes in question are the government's acquisition process and the contractor's development process. How these are carried out is a critical determinant of the risks and outcomes of product acquisition. The products in question are software and its accompanying documentation. Software products, however, do not stand alone. They have meaning only in the context of the digital hardware architecture that software causes to operate, and in the functional and performance requirements of the entire system in its operating environment. Both processes and products involve people -- multi-disciplinary technical and multi-faceted management people. The skill, experience, and communication ability of these people in both industry and government are critical to successful software development.

Underlying the software arena is a fast-changing digital hardware technology and a complex (and sometimes confusing) software technology that is rapidly evolving. These moving baselines for system technology and its implementation in vastly different applications continue to keep software on the cutting edge but also make it prone to technological risk.

Proposed strategies for dealing with current software development problems are presented in the following sections of this report. Section 3.2 discusses software development risks and some emerging technology that can reduce these risks. Section 3.3 discusses generic ways to improve the acquisition and development processes. Section 3.4 suggests ways to improve software products. Section 3.5 focuses on ways to retain people and improve their skills. Section 3.6 discusses, from the perspective of the technologist, software development technologies, both emergent and available, that can improve software development.

3.2 RISK REDUCTION OBJECTIVES

Risk reduction during software development is a vital contributor to procurement of complex, high-quality, software-intensive systems. Several technologies - tools and techniques - are emerging that offer considerable promise in reducing software development risks. The purpose of this study is to identify valuable emerging technologies and to formulate a set of recommendations that will bring these technologies into the mainstream of software development of Air Force Systems Command (AFSC) systems. Some of the software risk reduction objectives addressed directly or indirectly in this report are:

- Bridge the requirements design gap.
- Improve user involvement in the requirements process.
- Ensure continuity of design expertise.
- Design software for evolution.
- Improve latent error reduction.
- Tailor contracting to software characteristics.
- Increase investment in software technology and advanced development.

Software development of a large, complex, and unprecedented system is bristling with risks. An unprecedented system is a type or class of system not built before, or for which little or no design experience is available, or design principles or technology are poorly understood by the system designers; any of these factors magnify acquisition risk. A primary risk encountered early in development is translating a well-stated set of system-level requirements into a responsive software design. The ability to make this step skillfully and quickly is vital to successful software development. Many downstream difficulties and errors arise when this early step is done poorly or too slowly. Yet in 1988 no methodologies or useful formalisms existed for making this step; perhaps there never will be such automated aids for this purely intellectual task, although we foresee aids to test the merits of a design.

The leap from system-level requirements to software design is made by human beings who perform it more or less well in proportion to their creative spark. Furthermore, they must make their most critical software architectural leaps without benefit of science, mathematics, or much help from engineering principles. All system development teams need an "architect" who can successfully bridge this gap between systems phenomenology and software design architecture. When no such person is present, or when this essential gap is badly bridged, great tension arises between system design groups and software groups. In practice, a tactic for covering the absence of an adequate architecture is to schedule a lot of early (before preliminary design review) deliverables, giving an appearance, but not the reality of progress.

When it is recognized that this requirements-design gap cannot be bridged by methodologies or techniques but only by skilled, creative people, the search for solutions becomes clearer. These uncommon people must be supported by two classes of tools and techniques: (1) design exploration tools that experimentally evaluate the consequences of design decisions, and (2) support tools that relieve them from many tedious, mind-numbing tasks better handled by a computer. Both classes of tools are covered in this report, included in discussions on prototyping, iterative development, executable specifications, and other development issues. A major problem for AFSC is to make these needed tools an accepted and expected part of software development, as well as compliant with AFSC acquisition policies. At the same time, AFSC must avoid needless rigidity in its tool specifications, since the tools used must support software disciplines that are appropriate to the development program.

A second major software procurement problem that affects AFSC is the uncertain involvement with its user community. This problem is twofold: first, software is that part of the system that interfaces with the user, giving the system its perceived operational qualities. Design of these qualities is best done in conjunction with the user, starting long before system development begins and continuing throughout the development and testing phases. Second, when the procured system is transferred from AFSC to Air Force Logistics Command (AFLC) and the using command, a better hand-off is required than exists today. The present methods, which are based largely on formal transfer of documents and code, probably will not work if one-million-lines-of-code-plus systems must be rapidly modified in the depot or field to support post-1995 war fighting doctrines.

A third element in reducing software development risk is continuity and maintenance of technical understanding and expertise throughout the full life cycle of a system's development. This

period extends from the establishment of the requirements to the successful implementation of the operating code. Up to now maintenance of continuity and expertise has been attempted through a sequence of documents that, when stacked end to end, could be measured in tens of feet. No one was pleased with this process, but there were no better alternatives. Today there are better alternatives and the concept of transferring software design information through hard-copy documents should give way to more effective procedures using computers to search for and present desired information.

Designing software that can evolve gracefully far into the future is another critical concern for AFSC. Current software development methods, procurement approaches, financial reward systems, and political imperatives prevent building software able to reach beyond immediate goals. If there were real perceived value in building software for evolution into the future and if the needed technical qualities could be defined, implementation of these changes would greatly alter AFSC institutions. Although hardware may change greatly in kind as technology evolves, underlying system functions expressed through software often evolve more through addition than replacement. These conclusions suggest that, given portability to faster processors, software may become a more reasonable vehicle than hardware for long-term system evolution.

Due to their great complexity, software products have inevitably suffered from latent errors discovered in the field after formal delivery. The related initiatives of software quality, software reliability, software product integrity, and software safety address different perspectives of this problem. Current approaches are largely ad hoc, depending mainly on the creativity of the software designers. However, in contrast to the requirements-design gap intellectual problem, mathematically-based or logically-based formalisms are emerging that can improve our ability to rid systems of latent errors and enable increased quality, reliability, product integrity, and safety in software products. Section 3.6 discusses some of these topics.

Present methods for system contracting owe their origins to hardware-dominated procurements; generally they are poorly suited for many software procurements. For good results, software procurements need early phases in which design exploration, including exploration of ultimately bad ideas, is at least tolerated, if not encouraged; in later phases, single-minded, focused productization typical of the process described in DoD-STD-2167 can then be carried out. A demonstration/validation (Dem/Val) phase followed by full scale development (the process often selected for large weapons systems), is a good, high level generic model that can be adapted to meet the special needs of software development.

We want to emphasize that a software risk-reduction phase is appropriate, even for a software-based system not requiring hardware development or special integration. A planned preliminary phase prior to full-bore software development can represent the experimentation necessary to arrive at good technical solutions.

Where software development accompanies hardware development for an unprecedented system, the software risk-reduction phase should begin as soon as a system architecture has been proposed and should parallel the hardware risk-reduction work. Development of software typically does not begin until hardware has been fully specified. As a result, trades can be made in only one direction: asking the software to compensate for hardware or architectural

shortcomings.

Whatever processes and methods are adopted, it is likely that the cost per line of code will continue to increase. No method known to us will reduce costs of new code that must deal not only with today's problems but also with efficient parallel processing, secure processing, fault-tolerant processing, and many other new software, hardware, and system concepts that will appear. *If our recommendations are correct, and if they are implemented within AFSC software development, as development costs inevitably increase, the financial benefits will be derived through greater probability of timely and successful system development, longer useful system lifetimes, and fewer cost overruns.*

3.3 IMPROVING THE ACQUISITION AND DEVELOPMENT PROCESSES

Current Assessment

AFSC is participating in several initiatives that represent useful advances against software-related acquisition problems. The SEI is making progress in its development of techniques to evaluate contractor software development capability and to provide a needed software engineering curriculum for the country. The STARS program is moving to further use of Ada libraries and a "software first" development approach through its "competing prime contractor" contracts. DoD-STD2167A, the revised Software Development Standard, was released in 1988 and provides more flexibility in the acquisition approach. A guidebook to help acquisition organizations apply this standard will soon be prepared.

AFSC product divisions are also working hard to lessen risks in software developments. Every product division is moving to use Ada so that it can benefit from the increased discipline that Ada enforces. Aeronautical Systems Division (ASD) employs a Contractor Capability and Capacity Review during source selection. Furthermore, ASD has evolved a Software Integrity Program to provide disciplines for acquisition of software. Electronic Systems Division (ESD) has evolved Software Reporting Metrics, the Software Engineering Exercise (a contractor assessment test used during source selection), "greybeard" teams, red teams, (see Glossary) and a Software Requirements Audit to address contractors' requirements and management processes.

To be most beneficial, all of these efforts must be elements of a coherent management strategy which recognizes that many Air Force systems are on the leading edge of technology, where there are uncertainties and where one must continually trade off cost, schedule, and performance risks. These tradeoffs must be made in the context of a clearly understood acquisition process and a properly applied development model adapted to the needs of the particular development.

Below is a discussion of the familiar "waterfall model" for hardware and software development. Included in this presentation are the risks of using this model, comments on the development of requirements and how they drive the process, acquisition management issues, and life cycle software issues. Findings and recommendations are included.

3.3.1 The Waterfall Model

In acquiring systems with major software elements, the Air Force typically imposes a process referred to as a "waterfall model", for both hardware and software development during the full-scale engineering development (FSED) phase of the life cycle (See Figure 1). This process has yielded both successes and failures. Successes seem

to be characterized by three significant elements. These are:

1. a stable set of system and software development requirements that is not significantly different from that for a previously developed system or systems,

2. a digital system architecture and a software design that will satisfy the requirements,

3. contractor's systems engineering teams and software teams communicate effectively with each other and have had previous application experience in developing a similar system or systems.

A system with these three characteristics is a *precedented* system. Many of AFSC's procurements continue to apply the waterfall model to unprecedented systems not satisfying these criteria. Furthermore, these acquisitions are often established through firm fixed price contracts, geared to schedules which would only be reasonable for precedented systems. Schedule and manpower prediction models are often based on data for completed programs (both precedented and unprecented, but excluding failed and cancelled programs). Conclusions based on these models are sensitive to changes in input assumptions; since the inputs are uncertain for unprecedented systems, the credibility of predictions is poor.

Risk Reduction for Unprecedented Systems Using the Waterfall Model

The conventional waterfall model used during FSED usually assumes minimal risks. Therefore it does not adequately identify system risks and uncertainty, at least not early enough to trade and reduce these risks. Risks and unknowns in an unprecedented system development cannot be properly reflected in cost and schedule estimates without first taking the following steps:

1. Develop a team of systems, hardware, and software engineers who know the application area and the user's needs and who can communicate with each other so that each has an understanding sufficiently similar to work together on the same system.

2. Develop a digital system architecture and a software design that can illuminate the unknowns and identify the risks. For some systems, it may be necessary to test an implementation of the design in the real environment to permit adequate risk reduction.

3. Iterate the requirements and the design through trade studies and user inputs to establish validated, cost-beneficial, system and software requirements.

Given the satisfactory completion of these activities and adequate continuity of the engineering staff, one has a reasonable basis for cost and schedule estimation for subsequent, more formal implementation. The waterfall model then becomes an appropriate management concept for the remainder of the development process.

Some application areas in AFSC have little difficulty with the waterfall model, but only because they are strongly precedented. These include most training software (flight simulators), satellite system software, unit-under-test software for automatic test equipment, some air defense systems, and conventional surveillance systems. These classes of programs encounter serious difficulties only if requirements reach too far beyond those of previous systems or if too few application-experienced technical and management staff are available to the

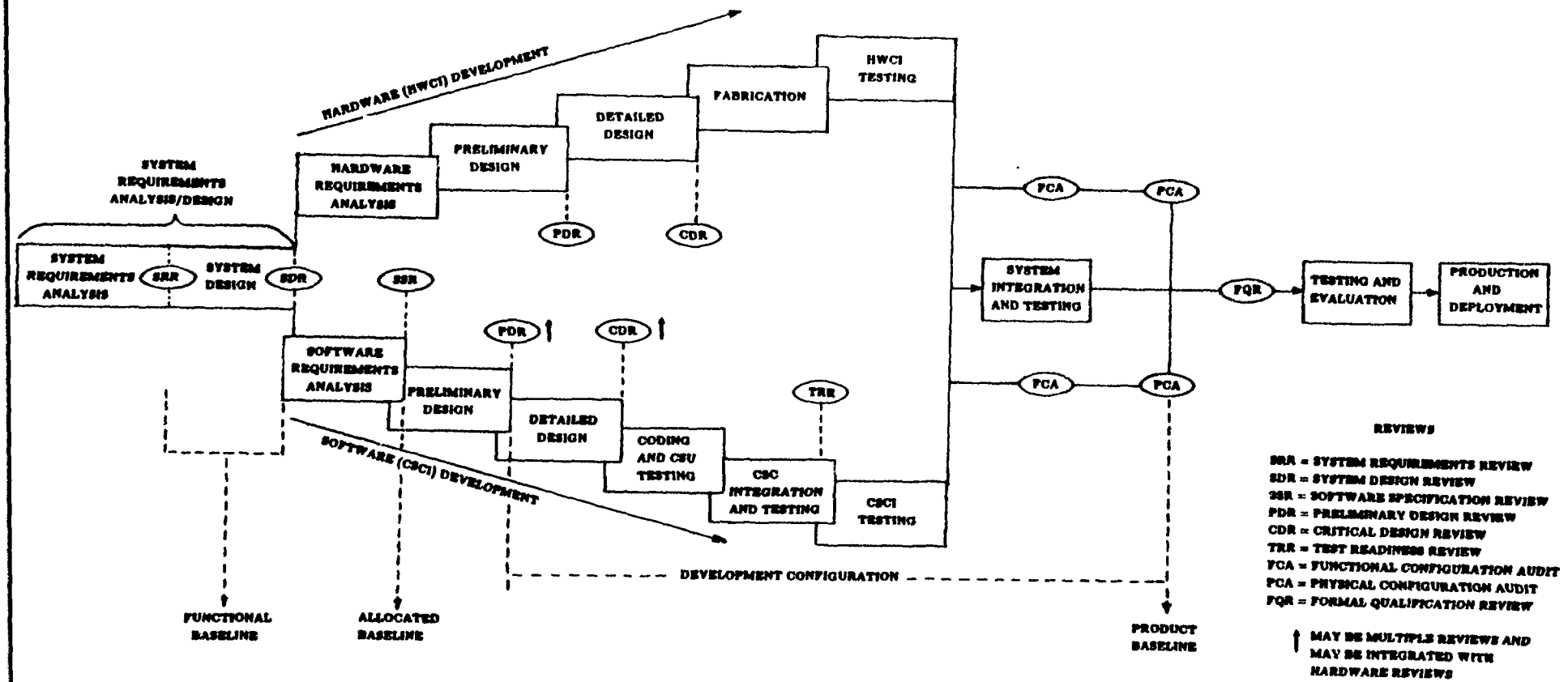


FIGURE 1 THE WATERFALL MODEL OF SOFTWARE DEVELOPMENT

program.

RECOMMENDATION 1: Air Force Systems Command, working with the Joint Logistics Commanders organization, should ensure that development models and accompanying rational alternatives to the waterfall model, based on risk reduction concepts, are included in the forthcoming Handbook 287 for DoD-STD-2167A, with supporting direction in AFR 800-2 and 800-14.

Other Development Models

Table 1 lists several system situations and suggested processes for development. For the models depicted in the handbook, approaches to reducing risk and trading off cost, schedule, and performance risks should be described. The presence of alternative models in the handbook should encourage tailoring of acquisitions so that risk elements are properly addressed. In addition to the handbook itself, AFSC acquisition personnel should have training in tailoring. Management oversight will be needed to ensure that intelligent compliance is taking place.

3.3.2 Development of Requirements and Design

Many applications in AFSC are unprecedented, either because a new system is so different that no one has done it before or because the particular participants (user, acquirer, contractor, managers technical staff) on the project have not done it before. Examples reviewed by the committee included the Strategic Air Command Digital Information Network, which had the first multi-level secure operating system; over the horizon/backscatter radar, which was the first large, distributed system with which the contractor team was involved; and the B-1B Defensive Avionics System, which also was the first large software development for its contractor. For such systems it is important to verify in advance that the requirements and design can be achieved within known and controllable risks.

Late and Risky Requirements Baseline

During FSED for unprecedented systems the Air Force contracts (often on a fixed-price basis) for software whose requirements and design risks have not yet been identified. Software development requirements are allocated and derived from operational requirements and system functional requirements only *after* contract award. The true cost of some requirements may be unknown. The contractor's digital systems engineering activity may propose an overall system architecture too ambitious to be feasible or affordable.

When hardware encounters difficulty, software is a favored work-around medium, because it is thought "flexible." The software must accommodate a poorly defined and continually changing mission environment. Overall, the software requirements baseline for unprecedented systems is completed too late in the program, without an adequate resolution of risks and without an adequate basis for estimating development cost and schedule.

RECOMMENDATION 2: AFSC should ensure adequate software risk reduction for unprecedented systems during a demonstration/validation (Dem/Val) or equivalent phase preceding full-scale development. For unprecedented systems, AFSC should provide policy guidance for competitive two-phase procurements, such that software risks are reduced to a practical minimum before proposals are prepared for the following phase(s). For unprecedented systems, AFSC should direct an independent assessment of system and software risks near the end of the Dem/Val phase

or Phase 1 of a two-phase procurement, as part of the basis for follow-on procurement decisions. For multiphased procurements, AFSC should direct that contract mechanisms provide continuity of essential contractor skill and expertise between contract phases.

Software requirements, risk identification, digital architecture adequacy, software design issues, and operating environment uncertainties should be resolved during the Dem/Val phase of acquisition. Digital system architecture and key parts of the software should be benchmarked and prototyped to validate the design and to assess its timing, memory sizing, and data bandwidth capability. In some cases a preliminary implementation of the complete real-time software component will be needed. Sometimes the users will need to be involved during these preliminary activities to ensure that the stated requirements truly represent their needs.

Where there is no Dem/Val phase, a competitive two-phase procurement (with down-selection at the end of the first phase) could be employed for unprecedented systems. Near the end of Dem/Val or first-phase procurement, an Air Force software systems engineering advisory team independent of the program office should review the effort to ensure that critical software issues have been addressed and that risks are known and acceptable; otherwise, the next phase should not commence.

It is important to maintain continuity of the contractor systems engineering and software engineering teams into the next, full-scale design phase. Many key personnel will be lost to other activities if the technical team remains unsupported or without meaningful activity during lengthy source selections. The Air Force should structure acquisitions so that key technical personnel are given contract support and provided useful continuing activity until the subsequent contract is signed. This continuity of knowledgeable staff is essential to risk reduction. One example of how this might be donewould be to define part of the FSED work to be accomplished during the Dem/Val contract, but *not* to be evaluated as part of the competition for FSED. This step would eliminate the confusion and schedule delay of restarting the program, at the expense of funding additional redundant efforts during the source selection interval.

RECOMMENDATION 3: For key technologies in systems and application areas where operational threats or requirements change rapidly, AFSC should fund parallel technology programs at the system level to foster a ready industrial base from which to compete single phase system acquisitions.

Changing Requirements

For applications such as electronic warfare, requirements continue to change rapidly, so that a system design based on early requirements may not be able to handle new ones. For such important applications areas it is necessary to maintain a warm industrial base by funding technology programs that look beyond the component level to the system level. Requirements and compatible designs for digital systems (including the software) may well be implemented, flight tested, and evolved by several competing companies. When a new or modified weapon system is needed, requirement-compatible designs and skilled design teams will be available to compete and extend their products for low risk implementation in systems. This technique is followed for jet engine development and could well apply to the digital system and software area for key technologies and application areas.

To summarize: AFSC can develop requirements and contractors can assess

Examples	Understanding of requirements	Applicable commercial software	System architecture understood?	Application criticality & risk	Future evolution understood?	Candidate process model
Inventory control resource monitor	Well understood	Complete application packages	Yes	Low	Yes	Acquire COTS
Updating of fielded system	Well understood	Probably none	Well understood	Low to moderate	Reasonably well	Waterfall (with Ada)
MIS portion of command & control system	Still converging	DBMS, prototyping tools, screen gen.	Usually not	Low to moderate	Generally but not in detail.	Evolutionary development (build, rebuild)
Expert system, intel. analysis tool	Still converging	AI shells	Yes, but with limited coupling	Moderate	Not at all	Exploratory, for limited field use.
Most C-cubed, avionics	Still converging	Prototyping tools, VHLLs (for evolution)	In part	Moderate to high	Generally but not in detail	Prototype, followed by waterfall
Large distributed real-time display-oriented systems	Only near-term understood	No	No	Moderate to high	Poorly	Incremental development

TABLE 1 SOME SYSTEM SITUATIONS AND SUGGESTED PROCESSES FOR DEVELOPMENT

the risks and impacts of these requirements in a single procurement phase for precedented systems. But for unprecedented systems or systems with rapidly changing requirements, requirements and designs must be verified and teams must be built before cost, schedule, and performance baselines are established. Otherwise AFSC should expect continued program slips, cost overruns, and performance shortfalls, if not outright failure of unprecedented system developments.

3.3.3 Management of Acquisition and Development

Advanced Development of Software During System Full-Scale Engineering Development

The committee observed that in too many cases the equivalent of advanced development is being pursued for software during FSED of the system. We believe that this results from three major causes:

- Lack of proper examination, experimentation, and prototyping during Concept Definition, Dem/Val, or other pre-FSED activity.
- Late discovery that some required functions intended to be implemented in hardware cannot be practically implemented and are shifted to software. This shift might not occur save for the prevalent optimistic view of the pliability of software. In truth, software is *not* pliable in large, complex systems. *A small change in a software function can ripple through many interfaces*, amounting to a major redesign effort, particularly if the added function was not anticipated during the decomposition of computer program configuration items (CPCIs) and modules.
- Changing requirements during acquisition. Requirements change because of changing threats, changing perception of threats, advancing technology, or because the user/operator has changed his views during early phases of the system acquisition.

The first of these items can be corrected by formally requiring early identification of Critical Software Issues (CSIs), development of a risk management plan, and adjustment of the plan at key milestones, carefully capturing the decision process and rationale for key software decisions. For example, critical software threads and modules should be identified as early as possible; risks should be quantified through study, experimentation, or prototyping. Some of the prototyping will be in the specific modules or threads; some will be at the subsystem level. In those cases where the program office does not have enough expertise to evaluate the risk management plan, a "nestful of owls" (Recommendation 15) can be employed to advise.

RECOMMENDATION 4: Each program involving software should be required to carry out early identification of critical software issues and to develop and maintain a Software Risk Management Plan. This instruction should be included in the most appropriate Air Force management directives.

Late transfer of function to software can be minimized by more expert system engineering at the front end of a program. Late discovery that hardware cannot practically perform all that we aspired for it is inevitable. When faced with the event, there are two major choices: shift the unfulfilled function to software and press on, or defer the unfulfilled function to a later time or capability block. Too often, the first choice seems to be an automatic response, without proper examination of

the consequences; the second choice may be much better in most cases.

Change of requirements cannot sensibly be prevented. As with other late-recognized situations, there should be a conscious, well-reasoned decision to pursue changes during the current acquisition, or alternatively to defer some or all of them to a future block.

RECOMMENDATION 5: When a program manager is faced with late identification of software requirements that can be deferred to a later time or capability block, AFSC management guidance should encourage and support this deferral and accept the consequences of doing so.

The software-intensive systems that DoD acquires are often intentionally specified at the leading edge of technology. Given ambitious aspirations it should not be surprising that the contractors do not always succeed completely in fulfilling them all. DoD and the Air Force should not indulge in self-flagellation when falling somewhat short of the goal, nor should they accept being beat about the head and ears by the traditional critics. Instead, they should make mature decisions, in light of the riskiness of leading edge endeavors, and articulate the reasons for such decisions simply yet convincingly to the Office of the Secretary of Defense, Office of Management and Budget, Congress, and the news media.

Inadequate Tailoring of Acquisitions

Both government and industry representatives told this committee that inadequate tailoring of acquisition processes and formal requirements often produces a bad starting position, especially for software-intensive programs.

In almost every program briefing, Air Force personnel cited a need for *more* user involvement. However, our observation is that the need is not always for more, but for *better structured* user involvement. For example, in the design and development of a radar system, much of the design is constrained by the laws of physics and the state of the art; it really does not matter much which specifics the user/operator prefers. But there are some operational features that should involve the user/operator. For example, the tradeoff between probability of detection and false alarm rate, the maintenance concept, and all man-machine interface aspects (e.g., tasking, reporting, and displays). In contrast, commander in chief command centers are today for the most part quite mature. The software is, in large measure, a manifestation of standard operating procedures for the command post staff. The software is probably maintained ("modified") by in-house staff. It is inconceivable that such a configuration could be significantly upgraded in a turnkey mode. The user/operator must not just be "involved"; he must, indeed, lead the major upgrade. We note that the Granite Sentry program seems to recognize this modality. (Granite Sentry is updating the strategic warning software at NORAD's Cheyenne Mountain in Colorado.)

We did not hear any government representative express a need for tailored user involvement. We believe that this missing perception is remarkable, that it should be corrected, but that it probably requires the help and counsel of especially wise and experienced staff not usually indigenous to a program office. We will discuss this resource later. Tailoring user involvement to each program is only one of many reasons why such an organization is needed.

RECOMMENDATION 6: User involvement should be tailored for each program, varying from cases requiring

very limited involvement to ones in which a user will assume the lead role.

We heard from both government and industry that the software engineering and development process is overspecified. DoD-STD-2167 was severely criticized; DoD-STD-2167A was cautiously applauded as an improvement over 2167, particularly in its permissiveness with the process. However, 2167A was still criticized for a lack of overt encouragement of innovative and tailored features of the process. Standards 2167A and 1521, taken together, still strongly discourage any approach but the waterfall model. When combined with early preliminary design review (PDR), this forces all program effort into the production of deliverable documents. Contractors cannot reduce risks because they cannot perform the necessary front-end exploration of unknowns.

We learned that DoD Handbook 287 is being prepared to augment 2167A. Such an aid is desperately needed, provided it properly addresses tailoring. However, we are concerned that the handbook alone will not be sufficient. At the very least, some change in what we regard as counter-motivation will be required to encourage people to practice what the handbook describes.

We note that an individual may work on perhaps four major programs during a career. The learning curve on tailoring that this affords is probably weak because from the individual's point of view, program starts may be five or more years apart; experience in one program may not even apply to one's next program. Hence, we are driven again to the notion of a group of software systems engineering advisors who can observe many programs over a relatively short time span, enjoy a rapid learning curve, and apply lessons learned immediately. The important result is that the contractor must operate under a well-defined, disciplined development process that he can practice effectively, not necessarily the one specific process illustrated in DoD-STD-2167.

In addition to inadequate tailoring of 2167 and 1521, other standards and specifications were reportedly also overspecified. Besides citing too many standards and specifications, a program office may insist upon some performance parameter although the impact on other system parameters may be unknown, adverse, or uncontrollable. Such nonanalytical requirements distort resource application, misfit designs in important parameters, distend schedules, and raise costs. The time to prevent inadequate tailoring of specifications is at the beginning of the program, not after it is recognized to be in serious difficulty. But given the scarcity of skilled system engineering personnel, the necessary tradeoffs may not be identified early by organic program personnel.

We were also told that when the Secretary of Defense demands something such as mandatory use of Ada, program offices are reluctant to process waiver requests no matter how sensible they prove to be. This reluctance becomes especially acute when program office personnel lack confidence in their own ability to judge the need or to persuade the decision chain. Both specifications and waivers are topics where expert skill and experience will help. However, one should not rely largely on "spec experts" whose careers are vested in the continuity and growth of a specification, but rather develop broader expertise that is better able to balance acquisition issues.

Firm Fixed Price Contracts Undesirable for Unprecented Systems

A firm fixed price (FFP) contract that has produced a poorly defined effort does *not* shift risk to the contractor. It merely increases the risk

that the program or some important part of it will be sacrificed. The decision of what to sacrifice comes late in the acquistion cycle, and is typically made by the contractor. The contractor presents the decision to the government as a *fait accompli*, leaving the government with few options, and no completed designs or systems.

Firm fixed price contracting of poorly defined efforts defers overt confrontation of trouble until "somebody else's watch," hardly a benefit to the Air Force or the taxpayer. In fact, the cost of software for major systems is still a small part of the total. It might make sense to develop system software under cost-type contracts with incentives for improved maintainability.

RECOMMENDATION 7: AFSC should direct its product divisions to tailor the contract form for each specific program's needs; in particular, AFSC should avoid using firm fixed price contracts for unprecedented programs. (This approach will require management followup, consistency, and the support of higher authorities.)

Mandate Use of Software Engineering Environments

Software engineering environments (SEEs) are integrated, computer-based collections of software, test data, programming, and management tools that focus the coordination, documentation, programming, testing, and project management for complex software developments. A representative set of capabilities of an SEE is as follows:

Minimum (kernel) - configuration management, requirements traceability, data base support, access to commercial packages, documentation support, programmer's notebook, compiler/-debugger, etc., platform, test vehicle, controlled remote access.

Design Exploration Set - design analysis tools, libraries of reusable code and simulators, very high level language support, syntax directed editor, application-oriented languages, support for executable specification, test effectiveness tools, quality assessment and correctness tools.

Management Support Set - acquisition process models, schedule analysis, action and error tracking, coupling to SOW, WBS, priorities, deliverables, personnel, metrics collector, early warning, acquisition management communications.

This list of capabilities is not intended to serve as a template. AFSC pamphlet 800-43 (p. 14) shows a more comprehensive list addressing the software development subset. We recognize the role of SEEs in supporting our recommendations. By its very nature an SEE will be the preferred platform for implementing many of them.

The value of SEEs in AFSC's future is difficult to quantify because no SEE is yet in sufficiently widespread use to prove the value of this important concept, although a large number of SEEs are in development, and ad hoc or first generation SEEs are now in use. The most serious effort, one that may serve as a model for others, is the procurement now under way by NASA for the space station. NASA made an important decision two years ago to improve its control of software procurements by furnishing a common SEE to its software suppliers and those of its centers involved in space station software. An interim system exists today, assembled from existing elements. The first version designed from scratch for space station software development is scheduled to be delivered in November 1989.

If we are to recommend a strategy for employing SEEs at AFSC in support of improved software development, the

following questions must be answered:

1. Are SEEs needed to introduce new software development technology?
2. Can SEEs support better management and acquisition techniques tailored to software developments?
3. How should AFSC motivate its software suppliers to use an SEE?
4. How should AFSC use SEEs in handing off systems to AFLC and to using commands such as Strategic Air Command and Tactical Air Command?
5. Should AFSC use SEEs within its own command, and if so, how are they best procured?

The remainder of this section tries to answer these questions.

1. Are SEEs needed to introduce new software development technology?

The design exploration techniques discussed in Section 3.6 of this report all require development platforms that are fast and provide large storage repositories for information. Furthermore, the software architects and designers using these exploration techniques all need immediate access and fast turnaround. These needed qualities characterize an SEE.

The alternative to an SEE is a loose collection of unintegrated tools. Each tool supports some isolated development function that is executed at some time during the life cycle. Design information or procedure changes for each tool must be introduced by hand throughout such a system without means to transfer information between tools. This kind of user interface smothers effective use of the new technologies discussed in Section 3.6.

2. Can SEEs support better management and acquisition techniques tailored to software developments?

SEEs can provide considerable electronic support to management and acquisition functions, blending into the process in a very natural way. A manager could audit program progress without interrupting the workers. Productivity metrics could be measured in this background without disturbing those who are producing. A manager's ability to ascertain true, current status on a weekly or even daily basis throughout a geographically dispersed development team would be unparalleled, compared to a non-SEE environment. The need for elaborate face-to-face design reviews, and extensive documentation typical of Standards 1521 and 2167A is much reduced, though not entirely eliminated. Electronic capture of a model of the software development process allows resource reallocation decisions to be based on real answers to "what-if" analyses of cost and schedule projections.

3. How should AFSC motivate its software suppliers to use an SEE?

Procurement instructions in a request for proposal (RFP) are the simplest and most direct way to require that contractors use SEEs. Furthermore, the process of evolving an SEE is an instructive area for comparing the software expertise of rival contractors.

The biggest problem in using an SEE is not a technical problem, but a cultural one. Although resistance by software engineers to automation is far less than that of other workers, some working software engineers resist moving into new ways of doing things. To ensure compliance, AFSC must first order that an SEE be used, then follow through with a pre-contractual exercise

in which the competing contractors are forced to use their selected SEE to produce code.

NASA, for example, has directed its software suppliers for the space station to use a NASA-procured SEE that NASA is furnishing to them. AFSC could do this, but we believe AFSC should instead direct and enforce the use of an SEE by its software suppliers and leave the choice of SEE to the supplier. Meanwhile, AFSC should watch closely the progress of NASA in its unique approach.

4. How should AFSC use SEEs in handing off systems to AFLC and to using commands such as Strategic Air Command and Tactical Air Command?

System hand-off of software today is primarily based on transfer of product code, explanatory documentation, computing hardware and training programs. The weakest link in handoff is the explanatory documentation that is at best infrequently used and at worst out of date and useless.

If system hand-off is based on transfer of an SEE and its associated stored information, the transfer process will be much improved. The retrieval of electronically stored information free from the inherent restrictions of documents and the printed page is probably the most significant advantage. The ability to change code, investigate the consequences of the change, and execute stored formal tests against expected output is made possible by an SEE. Self-training could be simplified and made more effective if based on an SEE.

5. Should AFSC use SEEs within its own command and, if so, how are they best procured?

First, AFSC can piggyback onto each contractor's SEE, system by system, without procuring a system of their own. Second, AFSC procurement organizations can selectively acquire the program management elements of an SEE appropriate to the organization and its system areas for use within AFSC's computer networks or on newly acquired mainframes.

In piggybacking, AFSC could buy terminals, modems, and communications security equipment and hook up remotely to each contractor's system. This is probably the least expensive approach, and has the twin virtues of heightened understanding of the contractor's development approach and a closer working relationship. The specification and regulation of the electronic interface between AFSC and its customers could be handled by means of the data item description (DID) mechanism currently used. The main limitation of this approach is that contractors will inevitably object to customers rooting about in their internal data, and probably set up separate data for AFSC access by terminal.

One potential drawback of using contractor SEEs is that AFSC and its product divisions would potentially use a different SEE for every contractor. Where this is too cumbersome, AFSC could standardize on a few SEEs for its internal use, and ask through DID specifications that each contractor transfer electronic information from their SEE to AFSC's SEE, in accordance with predefined requirements of the DID. In any case, this will leave the contractors free to define their chosen SEE in whatever way suits them, yet will adhere to AFSC standards for electronic information transfer.

There are some relatively simple ways for AFSC to improve upon the current situation:

- Specify an SEE that supports Ada, appropriate software development processes and methods, and includes:
  - configuration management tools (including bug status tracker),
  - requirement traceability tools,
  - documentation generator,
  - compilers, debuggers, loaders, linkers, and support subroutines,
  - unit development folders (or similar features that support the development process to be used).
- Make sure that all of the above are compatible or integrated and proven, and that staff are trained in their use.
- Require a positive response to the specifications from bidders, in their proposals.
- Include an audit of the SEE during precontract award audits.
- Require an SEE in order to qualify a contractor.

RECOMMENDATION 8: Product divisions should be directed to specify use of an SEE for each program having, as an example, a software staff of more than 12 people, and to require proof of its existence and the contractor's knowledge of its effective use, to qualify.

Software Engineering in a Distributed Collaborative Environment

As software-intensive systems become more complex, the team becomes more widely distributed. System complexity demands tight communications linkages, but physical or organizational separation tends to weaken communication.

The problem manifests itself through divergence in system understanding, interface mismatches, and general management control difficulty.

There are two major short-term fixes:

- Use modern, electronic communication, for example, video conferencing (where the mass of the software force can justify the cost), electronic mail, distributed data bases, and the like.
- Where practicable, divide tasks to minimize the required interface between physically and organizationally disparate entities.

The longer-term solution is to develop a software management environment designed for distributed efforts using new high-speed networks.

RECOMMENDATION 9: When software development contracts are granted to design groups that are organizationally or geographically separated, near-term management criteria in source selection should emphasize use of modern telecommunications and division of tasks to reduce requirements for interface among separate locations or organizations. In the long run, contractors should demonstrate availability and use of a software management and engineering environment designed for such dispersed efforts.

3.3.4 Software Life Cycle Issues

Later life cycle issues follow the acquisition of systems containing software. Such issues include software support and maintenance, incremental development of replacement software, and the various roles of software documentation in these processes. For many automated systems, AFLC provides the software maintenance service while

for others the user provides the support. Whoever supports a system must have adequate tools to maintain it.

Computer Resources Working Group (CRWG) Activity

Users and operators, under promise of nonattribution, told us that some computer resources working groups (CRWGs) are mere facades. In some cases, their activities are pro forma; they meet at key milestones but accomplish little. In a few cases, we were told, CRWGs never met after the initial organization meeting.

We believe the CRWG is vital to the support of software development and maintenance processes. Program/project managers and software acquisition managers must become aware of the importance of CRWGs and insist that they perform their necessary functions.

RECOMMENDATION 10: Product divisions or Headquarters AFSC should regularly monitor CRWG performance. Explicit evaluations should be solicited from using commands and AFLC.

Documentation for Maintenance

An important and often expensive life cycle issue is: what documentation is truly necessary for maintenance. Some documentation *purchased* for maintenance is not used for maintenance, wasting time and money. Some documentation *needed* by maintenance may not be provided, or may be provided late. When a program is cut back, maintenance documentation may not be included.

RECOMMENDATION 11: AFSC, with AFLC and the using commands, should sponsor a fresh look at actual maintainer documentation needs. This review should include consideration of the growing automation of documentation by contractors, and how that might be used to reduce the cost or improve the utility of the data.

AFSC, AFLC, and the users should jointly examine needed maintenance documentation, not just for new starts, but for ongoing programs where documentation and associated tool requirements on contract are subject to change. When programs are curtailed, AFSC must recognize the need to protect completion of support environment and tools, and act accordingly. Software documents should be comprehensive and designed for their users, at least to the degree to which this is done for hardware, yet should recognize the difference in the two kinds of maintenance personnel and functions.

Incremental and Evolutionary Development

Incremental development and evolutionary development pose reasonable approaches to the software life cycle for some systems. Briefers expressed some concern that incremental or evolutionary development may not succeed because the Office of Management and Budget (OMB) or Congress would not support the concept. These fears apparently are partly based on known attitudes of some principals or staffers and are partly conditioned by past experiences.

We believe that lack of support may result from poor articulation of the need for and benefits of incremental and evolutionary development in some systems. After all, the OMB and Congress already support evolutionary incremental developments: block programs (AWACS, space programs, etc.) and modification programs. Hence, no fundamental conceptual barrier to funding of such programs should exist. If someone does not want to do something, for whatever reason, then one excuse is as good as the next.

RECOMMENDATION 12: AFSC should select an appropriate program (or programs) through which to implement incremental acquisition, using it (or them) to articulate to OMB and Congress the need for and special benefits of an evolutionary, incremental, acquisition approach.

A key here is to explain to the principal parties the benefits of incremental or evolutionary development. But the Air Force must also recognize and accept rejection, when the rejection for incremental or evolutionary reasons is really a red herring. Discover the real reason, and work on that.

User and Logistics Command Software Support

In many systems today, both the user and AFLC claim roles in software support. The user maintains that he needs more responsive modification capability for a wartime scenario, while AFLC maintains it has the skill, facilities, experience, and charter. The result could lead to maintenance of the software at two separate sites. This concept is not viable or cost-effective if the software is highly integrated or highly interactive. The success of user modification of software will depend on availability of a complete development environment in theater. In most cases it will be practical if theater changes are limited only to changes of selected parameters by some sort of reliable parameter-switching feature in the software. The central maintenance activity that performs actual software modifications should include knowledgeable using-command liason officers who have interfaces to in-theater users.

RECOMMENDATION 13: AFSC must strongly encourage AFLC and the using commands toward collocated support for software in integrated systems, rather than complex reprogramming without adequate resources in the field.

AFSC must make clear that with today's technology the risks are very great when integrated software is maintained at separate sites or by separate organizations. Multisite modification is reasonable only if particular software can be designed to be only loosely interactive with other software and subsystems. It is far better to limit field changes to parameter changes tested extensively before fielding. Perhaps if future technology permits the communication bandwidth between two sites to be large and reliable enough to approximate that of collocation, multisite maintenance may become feasible.

3.4 STRENGTHENING PERSONNEL RESOURCES

The process of acquisition for software-intensive systems often gets off to a poor start. Even before release of a solicitation, the process is overspecified. Large numbers of standards and specifications are being invoked without serious justification. The tailoring process, encouraged by DoD-STD-2167A, is often not followed by the AFSC product division, but by the bidders.

Two major reasons for this shift of the onus from government to contractor are (1) lack of confidence by the government personnel in their own knowledge and capability to tailor and (2) perverse motivation. If tailoring cannot be done intelligently, rigid adherence to every requirement may be "safest." There is no reward for attempting tailoring that turns out badly, and no penalty for failing to tailor or for overspecifying standards and other constraints.

Bidders are reluctant to recommend serious tailoring, because they believe

this could jeopardize their chance of winning the bid (either because the government evaluators may not recognize a good tailoring recommendation or may not agree with it). Further, during the course of acquisition, the AFSC program office may be reluctant to process necessary waivers (e.g., an Ada waiver) because they are not confident of their own judgement, not confident of their ability to persuade the necessary decision authorities, or concerned about the time delays added by these actions.

The people who can competently and confidently exercise judgment in each of these cases are system engineers with strong software backgrounds, or software engineers with broad understanding of systems. Each is in short supply; indeed, given the constraints on personnel management resulting from overall policies and public law, there will likely never be enough of these types. This suggests the following course of action:

- Try to widen the base of such personnel.
- Prepare and use guidebooks that are compact and easy to use even by moderately skilled people.
- Through special management, use available personnel to best advantage.

RECOMMENDATION 14: AFSC should broaden the base of its personnel skilled in acquisition of software-intensive systems; prepare, use, and maintain current guidebooks; and exercise special management of the skilled personnel resource.

Widening the base of software-knowledgeable personnel must take in both military and civil service personnel, and will be carried out principally through widening the acquisition and education and training, as discussed below.

A handbook for tailoring DoD-STD-2167A is being prepared; its early publication should be encouraged, and its use fostered. In addition, guides for Ada waivers and for application of standards and specifications, in general, should be prepared and used.

Special management, in general, requires some measure of centralization of management of the scarce resource. For example, in the short term at least, a software systems engineering advisory team could be assembled to advise on tailoring DoD-STD-2167A to produce an acquisition process for a specific program. Advisory teams can be formed inside a product division if enough skilled people can be brought together. Cross-divisional or command-wide coordination may be necessary, if skilled personnel prove to be scarce.

This "nest of owls" concept was used successfully during the 1970s in the Air Force MIS ADP (management of information systems automatic data processing) community. Eighteen professionals were assembled to "fight fires" (to help the Air Force recover from software debacles). They were soon redirected toward "fire prevention" (advising at critical stages in system development, beginning early in the cycle). The charter was drawn carefully to prevent infringement on the authority of the program manager; the function was to advise both the program manager and higher levels of management. So that the group would not be regarded as an inspection group or auditing group, the detailed report was provided only to the program manager; a highly abstracted version of the report was sent to higher management. Where remedial action was required that was beyond the resources of the program manager, recommendations were made directly to higher management.

RECOMMENDATION 15: AFSC special management of software skills should include a software systems engineering advisory team (a "nest of owls") and special career tailoring for selected officers and civilians.

Product division commanders must first understand that the present motivation for tailoring is counterproductive, and then change the motivation to favor innovative tailoring in their own divisions and by industry.

Policy statements might be a good start but other steps certainly will be required:

- workshops in or across divisions, workshops with participants from industry, codification of their results in guidebooks and in model paragraphs for solicitations;
- diagrams that show how prototyping (of several varieties) can be accommodated within the workflow.
- modified DIDs that fit the modified workflow diagrams; and
- modified work breakdown structures that fit the modified workflow diagrams.

RECOMMENDATION 16: AFSC should take steps to increase the motivation for innovative acquisition tailoring. AFSC should issue a policy statement, conduct workshops, and distribute guidebooks.

3.4.1 Military Officers

Personnel management of officer participants in the software acquisition tasks within AFSC is virtually impossible to evaluate with the present state of data. We were told that some 320 officer positions within AFSC in the 262X, 273X, and 288X Air Force specialty codes are involved in system acquisition. However, we were also told that many officers in other specialty codes try to hide software acquisition experience because they see it as career limiting.

We were told that the average software acquisition officer would aspire to attain the grade of colonel at about the 22-year point, so that he or she could retire at a grade 0-6 at the 24-year point. We were also told that the typical software acquisition officer could not see a clear career path, even to lieutenant colonel; hence, we should not be surprised to see him or her resign early for lucrative work in industry.

A similar problem persists in the 49XX career field. Officers who are computer/software specialists cannot see a clear path to the top without career broadening. In the case of 49XXs, required career broadening must probably include communications management and air traffic control management, as a minimum. But the career broadening does not imply that they cannot serve on the computer side as they attain higher grades; it simply demands that they be capable of a wider range of assignments, or of assignments that include communications or air traffic control as well as computer resources. AFSC should broaden the officers, through assignment or training, so that they do not remain so highly specialized as they approach the primary zone for the rank of lieutenant colonel.

During the 1970s when the 30XX career field needed significant upgrading and motivation, a successful special monitoring program was established. Officers with high promise were identified, beginning at the senior Captain level. Their assignments were hand-tooled (not by the Military Personnel Center, which is very good at routine assignment though not at special assignment, but by the commander of Air Force Communications Command in col-

laboration with the senior Air Force communications officer in the Pentagon); each job and school assignment was carefully built on previous assignments and was intentionally more challenging than the last. Some officers failed to deliver on their promise; they were dropped from the program. Other officers fulfilled all expectations and were rapidly moved to the top of the field, both in assignments and in attained grade. The career field was significantly upgraded in about ten years, so that special monitoring did not need to be continued at such a high level.

As with any group whose members are chosen solely on the basis of merit, some aspects of a special monitoring program are not popular. Those officers excluded from the program will inevitably learn of it, will learn that they have not been chosen, and may become discouraged. The group might not strike a suitable ethnic and gender balance. And some of those who succeed in the program may still bail out early. Before such a program is adopted, the leaders must be willing to accept the negative as well as positive consequences.

There is no mid-career education or training course for software acquisition officers; nor, for that matter, is there one for system engineering officers, most of whom now should have a strong software background. A mid-career course at the point when an officer is ready to head a system program office (SPO) division would be very beneficial. Neither academia nor inservice schools offer a curriculum that fits the need.

Several actions that the AFSC could take to improve this situation are:

- In collaboration with the SEI, Joint Logistic Commanders (JLCs), and industry, persuade the leading universities to design an M.S. curriculum (probably designated as Master of Science in Software Engineering, and normalized through SEI) and to offer such courses in residence (for attendance through the Air Force Institute of Technology or other service equivalents) and for off-duty study (on consecutive weekends or at night).

- Work with the Defense Intelligence College and Defense Systems Management College to add courses or course elements relevant to system engineering with a strong software emphasis or software engineering with a broad system engineering foundation. Motivate the appropriate officers to take the courses, then assign them to positions where they can practice what they have learned.

- Conduct a course for each person involved in system acquisition along the lines of the Bold Stroke program (a computer indoctrination program for upper Air Force management).

- Develop and use a digital system acquisition business game, much as war games are used for teaching military strategy and tactics.

RECOMMENDATION 17: AFSC, in collaboration with others, should make available to officers and civilians a mid-career systems engineering/software engineering graduate program and appropriate short courses.

With relatively little effort and cost, AFSC can do much to improve the base of candidates for critical software acquisition management, to manage this critically scarce resource, and, over time, to diminish its scarcity.

3.4.2 AFSC Civilians

The situation for AFSC civilians

who are involved in software acquisition is much the same as for the AFSC officers, although there are differences. Although government civilians cannot retire after 20 years of service, they can (and some of the best do) leave early for more highly remunerative careers. Many seem to believe that software position classification may limit their careers to GM-13, no matter how long they remain in the field. This concern, combined with the relatively low civil service pay scales compared to industry, creates problems in building and retaining competent civil servants in the AFSC software acquisition business.

The flexibility that may be available for officer assignments is not generally available for, nor desired by, civil servants or civilians in industrial employment because flexibility implies movement. Most civil servants, like most industrial workers, eschew on-the-job moves. Nevertheless, within an AFSC product division there should be some flexibility to move people among SPOs and among deputy chiefs of staff (DCSs).

Upgrading the career field, in terms of pay level versus responsibility, is a governmentwide problem and probably requires a governmentwide solution. If the Joint Logistics Commanders are not already working with other government entities to push for upgrading, such activity certainly seems warranted. The problem is a general one, and there is precedent for the federal government to deal specially with critical skills.

The education and training issues seem virtually identical for officers and civilians, insofar as our recommendations are concerned. We believe both groups will benefit from the courses we recommend.

AFSC can improve the civilian base of candidates for critical software acquisition management, but probably at greater cost and with greater effort than is required for equivalent improvement in the officer base. Improving the civilian base is important. Civil servants generally provide a longer and more specialized corporate memory than the military, who are relatively transient. For the "nest of owls" concept, it will be very desirable to include senior civil servants who are intelligent and have valuable experience in software acquisition and system engineering. We suggest that individuals be utilized on the basis of their special skills and experience rather than arbitrarily grouped into specialist categories.

3.5 QUALITY OF THE SOFTWARE PRODUCT -- ACQUISITION ASPECTS

Product quality is only one of several interrelated software characteristics. Quality of the product and testability of software specifications are closely related. Furthermore, the quality of software should not be judged solely with respect to its performance, but also with respect to other quality factors such as maintainability (for software, maintainability means modification), correctness, and robustness in changing system environments.

Although there are some similarities of quality and testing between hardware and software, differences are more pronounced. For example, hardware and systems have a well-defined reliability discipline; reliability and availability equations are well understood. But for software, there is no wearout or breakage in the hardware sense; therefore there are no commonly accepted equations for defining reliability or availability. In fact, when software reliability is cited, it really means "reliability of the software development process."

Quality of software can be discussed in two distinctly different ways.

The first way is qualitative: either the software does its job so that the system operates as the user intended, or it does not. The qualities of a given software product are often erratic: some functions work well, others less so. The qualitative aspects of software involve (1) capturing the operational requirements, (2) translating them into functional requirements, (3) allocating the requirements properly between hardware and software, and to CPCIs and modules, (4) developing the software specifications with fidelity, and (5) implementing the software specifications with fidelity. Testing and operational use ultimately demonstrate the quality of the software performance.

The other face of software quality is quantitative. One would like to have metrics that can quantitatively show how well the software fits functionally and how many latent errors exist. Some of those metrics are described in AFSC Pamphlet 800-14, "Software Quality Indicators." Unfortunately, quantitative treatment of software is still in an embryonic research state. A few tools offered in the marketplace claim to treat software quality quantitatively, but these are neither widely accepted nor broadly used.

A research and development (R&D) program for quantitative evaluation of software can be defined, at least for technology pursuits. We address some software quality opportunity areas in Section 3.6 of this report.

System warranties are sometimes used to motivate the contractor to produce only high-quality software. The warranty usually covers all latent errors for the period of the warranty. Such warranties may have limited value, especially if the government modifies the software during the warranty period. Because modifications could introduce latent errors, attribution of any discovered latent errors can become quite difficult.

A better plan may be to require that development, test, and evaluation (DT&E) include explicit testing of each instruction (or better, each arc) in the code. All errors discovered in this way should of course be corrected before acceptance. Since the committee did not have the time and resources to analyze the cost and benefits of such a requirement, these factors should be examined carefully before implementing such a requirement.

RECOMMENDATION 18: The Air Force should consider revision of AFR 800-14, paragraph 5-3, Test Planning, and all derivative directives, to require demonstration of testing of every instruction within the software prior to completion of development, test, and evaluation (DT&E). Implementation needs, cost, and expected benefits should be analyzed by experts prior to implementing this revision.

Some software maintainers (ALCs and users) suggested IV&V by the organizational entity that is charged with maintaining the software after transfer of responsibility. These presenters seemed unaware that Air Force Regulation 800-14/AFSC/AFLC Supplement 1, paragraph 3-9, dated September 14, 1987, lays out just such an assignment. Nor did we hear anything from AFSC product division presenters that would lead us to believe that they were aware of AFSC/AFLC Supplement 1.

RECOMMENDATION 19: Each program should consider using the designated software "maintainer" (operational phase) as the IV&V agent during software development.

The advantage here is that the maintainer receives in-depth experience with the software much earlier than is otherwise likely.

We reviewed DoD-STD-2168, dated April 25, 1988 and have mixed comments. On the positive side, the DoD recognized the need for improvement in software quality and tried to do something about it. Unfortunately, this standard is wholly documentation-oriented, merely a checklist at a high level of abstraction. The quality program, moreover, is not discussed qualitatively in the standard.

We understand that a companion directive -- a handbook for government personnel similar in intent to that for 2167A -- is planned, and will be produced on a schedule paralleling that for the 2167A handbook. A software quality guide is desperately needed to instruct government personnel in how to evaluate a contractor's software quality program, and how to measure contractor progress against the contractor's plan. Experience will dictate frequent revision of this and other software handbooks, particularly if there is a clearinghouse to collect the results of experience. AFSC must support and encourage revisions to the handbook.

RECOMMENDATION 20: AFSC, with the Joint Logistics Commanders, should expedite preparation and distribution of the 2168 guidebook and support maintenance of this and other software guidebooks over time.

3.6 TECHNOLOGY

3.6.1 The "Software Problem" From a Software Technology Viewpoint

A possible software technology strategy for the AFSC would be to capitalize on anticipated software technology advances by other organizations, for example the DoD program duo, STARS and SEI. NASA's SEE, the industry-sponsored Software Productivity Consortium, and mission control complex programs are highly visible funded software engineering R&D efforts. Other efforts by industry, academia, and government also try to advance the state of software technology. A "watch and wait" approach by AFSC may be deemed appropriate in some engineering and scientific fields serving large commercial product markets, but is not appropriate here. The increasing complexity and quantity of Air Force software would result in technological stagnation, with respect to high-cost development efforts. Thus, in our view, the Air Force should take a broad and leading role in advancing software technology.

Technology should be used to attack the most worrisome software development problems that occur in large or high-risk projects. The rationale for pursuing these developments is to give the Air Force a world-competitive edge in advanced tactical and strategic systems. Large, complex software systems are the only way to achieve this competitive edge. Small, simple software systems are far easier to produce, but are inconsistent with the distributed, real-time nature of Air Force application domains.

The causes of large or complex software development problems are the software product, personnel, and their differing viewpoints. Complexity is high because the system is large or its internal connections are intricate, or both. The result is a large development team, consisting of system and software development groups, plus the Air Force, which in turn consists of the buyer, user, tester, verifier or validator, and maintainer or enhancer. Large team size implies increased team communications requirements.

Other problems include the fact that team members are not co-located and continuing problems with turnover (and lack of continuity of expertise). These factors set back the learning curve and produce gaps in corporate memory. In some quarters, software is

considered unimportant, or may serve handily as a scapegoat if systems encounter problems. This factor is compounded by a lack of promising career opportunities for software specialists.

These factors lead to major problems in development of large systems. A team simply cannot master all the requirements and other implementation needs. Geographic and organizational distribution of team members produces gaps in communications, interfaces, requirements, and understanding. These problems cause project disasters, e.g., software systems that cannot adapt to threat changes, fail to meet performance requirements, provide awkward user interfaces, and are poorly served by the hardware on which they must operate.

3.6.2 Modern Technology: Definitions and Descriptions

Unlike hardware technology, much of which is delivered to the system developer in ready-made materials or components, software technology has elements that must be "re-invented" and adapted to each particular system. Although software technology contains both application-enabling parts and process-supporting parts, the latter are of greater importance for this report. This production, integration, testing, and management technology does not reside in the fielded system but is required to build and enhance the system. (Resource limitations in the operational environment are a major reason for this separation of the product from the process support technology.) Programming languages and associated programming tools are the best-known vehicles of software production technology, but there are others, and more are on the way.

Although we accepted our subject without argument, we did not exactly agree on what constitutes "modern software technology." This is evidence of the current nature and breadth of the software field. Below, we discuss some of the elements of modern software technology and their known and potential roles in software development. In this section, we do not make specific recommendations that the subject technologies should be funded at some particular level. The right technology base program requires careful balance and competition of technologies serving the same purposes. We will not propose a technology strategy, but will discuss some of the technology that should appear in the Air Force's software technology portfolio.

Languages

Higher-level Languages, Specifications, and Prototyping. During the last 20 years many Higher Level Languages (HLLs) have become widely available. These include ALGOL, PASCAL, LISP, Ada, Modula, C, and Prolog, as well as FORTRAN and COBOL. HLL's do not require the programmer to include all the low-level detail that must be supplied in an equivalent assembly language program. Such unnecessary detail typically includes how arguments are passed to programs, subprograms are invoked, and registers are allocated. With an HLL, the compiler does the work of adding in low-level detail by making the necessary decisions and translating the results into assembly code. Importantly, most HLLs use familiar mathematical and logical constructs to simplify their use by programmers and others with mathematical backgrounds.

Performance, Correctness, and Utility. A *compiler* translates the programmer's work from HLL form into machine language. Usually compiler-generated code performs acceptably; that is, the compiler typically finds some but not all

of the optimizations that good human programmers find. For time-critical real-time applications, however, compiler-produced code often has fallen unacceptably short in performance. The principal exception is the language C, which performs better but is also at a somewhat lower level than most HLLs. Regarding correctness, compilers can be verified, validated, and tested so that they reliably produce code that is a faithful translation of the programmer's HLL program. However, there are often valid reasons for occasional use of assembly language, such as access to some hardware feature or external device, or time-critical performance not attainable using compiled code. HLL programs often perform better than assembly language programs for the same level of programming effort expended. More significantly, the HLL is more like natural language or mathematics. An HLL lets the programmer explore the application design space rapidly, and significant speedups often occur by searching for and finding better algorithms, rather than refining low-level code. Important exceptions will probably continue to be dealt with using assembly language, although *application-oriented languages* suggest an alternate route.

Re-usability. Re-usability is a broad term used in programming. It means that skills, program fragments, tools, techniques, and methods that have been developed for one application are put to work on others. At present, re-usability is a topic of much interest in software circles. Some technologists argue that the best way to re-use is to supply tools that permit a program to be built up from parts stored in a library, while others believe that the best way to re-use to employ powerful tools, such as compilers, which simplify programming. The success of re-usability depends largely on how far the re-user is from the user, organizationally and geographically. Individuals often re-use large parts of programs they wrote earlier, and most serious software engineers make extensive use of the available time-saving tools.

By moving up the chain to higher-level languages, we should gain greater reusablity. Since each step up can eliminate unnecessary premature commitment to detail (which is filled in later by the compiler), such as the particular hardware's instruction format, number, and type of registers, it becomes increasingly likely that larger portions of a program will be made applicable by moving them over to a new processor or problem. This advantage is illustrated in Figure 2. Informal re-use is of course difficult to quantify. Commercial and defense software contractors re-use parts of programs routinely. Rather than charging the government customer for "new" programming, the contractor most often uses the savings as a means to be more competitive. Savings through re-usability are concentrated largely in the task of *coding*; most other tasks in software development are only modestly influenced by reuse. Furthermore, since the programming process baseline already includes significant amounts of re-use, any demonstrations of expected savings must be examined closely.

Very High Level Languages. The next step up in sophistication from an HLL is a Very High Level Language (VHLL). These languages require still less program detail, usually performing their own data structure and procedure optimizations. Ada compilers, on the other hand, insist that the programmer provide them with details not required by many contemporary languages or VHLLs. Hence, in an integrated development environment coupled through an SEE, programmers using VHLLs should be less constrained than with Ada.

VHLLs are not only less complex than HLL's, but they are easier for humans to understand. Perhaps the

earliest general purpose VHLL was SETL, based on set theory, a branch of mathematics. Many consider APL (A Programming Language, developed by Kenneth Iverson in the 1960s and described in his book of that title) to be a VHLL, targeted on array operations. Another familiar example is more problem-specific in the data base area: a relational data base *query language* such as SQL can be both relatively simple but very high-level, the latter because most data structure and algorithmic operations are left to the discretion of the data base retrieval program. In practice this poses a performance issue for fielded systems but is satisfactory for prototypes and many typical data base problems.

Wide-Spectrum Languages. Some software technologists believe that wide-spectrum languages (WSLs) are the next evolutionary step beyond VHLLs. Wide-spectrum languages incorporate more than a single level of detail or abstraction. It would be consistent for a WSL to include not only powerful array of set-theoretical constructs, but also more primitive instructions providing access more directly to the computer hardware. This is one way of addressing the performance gap between machine languages and HLLs or VHLLs. Although modern HLLs may have access to "in-line machine code" inserted within an HLL program, this is a primitive capability that may locally disable the powerful consistency checking capabilities of the HLL compiler.

Unfortunately, WSLs will be complex. Experience since 1960 with IBM's PL/I language and more recently with Ada demonstrate the rule that "larger" languages require more complex and therefore more expensive compilers. Since the progress we make in software technology should also simplify the compiler design problem, WSLs may eventually become practicable.

Compilers and Program Synthesizers. As we move "up" the language spectrum to ever simpler languages (simpler for the human user), compilers must do more work to fill in the details from which the programmer is excused. Progress is limited by how well we are able to compile the more abstract levels of any language. The higher the language level, the larger the search space that must be explored by the compiler in search of a satisfactory program result. For example, whereas many of today's optimizing compilers search the programmer's work to find simple mathematical relations that are used repeatedly by the programmer, more sophisticated optimization will mean more complex searching. As our compiler or program synthesis technology improves, so will the simplicity of the VHLLs that can be compiled effectively into efficient machine code.

VHLL Executable Specifications - An Anecdote: The Ada-Ed Experience. Perhaps the best known VHLL experience is "Ada-Ed". Ada-Ed is a compiler for the Ada language, written in SETL (set theoretical programming language), a general-purpose VHLL that has sets, sequences, mappings and other mathematical or logical abstract objects as constructs. Those who know and use SETL believe it is simpler and easier to understand than an equivalent Ada program. Such a bias is, however, common among converts to any capable programming language.

The first Ada definition was ambiguous. The first Ada compiler written in SETL was written to simply state the rules of Ada, in an unambiguous form (though not in the highly optimized form necessary to an efficient compiler). For many implementors Ada-Ed really served as a first unambiguous definition of Ada. Although Ada-Ed is a "program" as well as a description, it was easy to read and understand. Also, it removed ambiguity by providing specific interpretations of

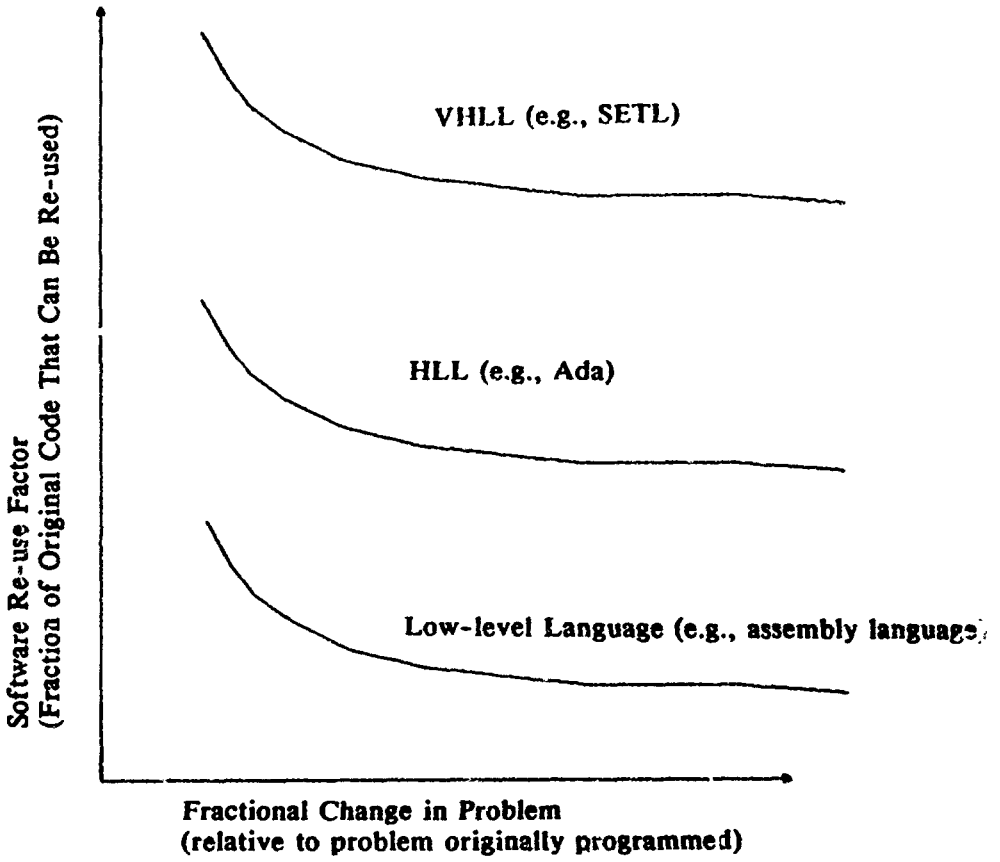


FIGURE 2 RELATIONSHIP BETWEEN SOFTWARE RE-USE FACTOR AND FRACTIONAL CHANGE IN PROBLEM

Ada operations. It could even execute programs, so that users could observe directly the results of test program cases. It was, in short, a prototyping tool.

A typical misevaluation of Ada-Ed would be to say that it was too slow. An inherent danger in having an operating but *non*-fieldable prototype is to miss the point of this prototype (which was to resolve ambiguities and communicate an unambiguous Ada specification). If the Ada ambiguities discovered using Ada-Ed had not been resolved before building later Ada compilers, those compilers certainly would have contained a proliferation of interpretations, and would have been more difficult and costly to build. It would have been more expensive to discover too late these ambiguities and perhaps attempt to patch fielded software written in Ada.

Technology to Narrow the Programmer-Computer Gap. By moving the software language closer to the user's own way of conceptualizing applications, the communication gap between programmer and computer narrows. The higher the level of the language and the closer it is to the user's way of thinking, the easier programming becomes. Personal computer spreadsheet programs may be the clearest example of meeting the user in his own bailiwick. VHLLs and their compilers form a most important technology to help in this area. Two other important technologies are specification languages and prototyping tools.

Specifications form the medium of communication among system or software buyers, users, builders, and maintainers. Ideally, specifications should be unambiguous, easy to understand and produce, easy to verify and validate, and easy to translate into implementations. Formal software specification languages have improved as we develop new specification methods, gain experience, and better understand the roles of specification languages. Language improvements include becoming more natural to the user, and coming closer to the constructs used in the application domain. One recent example of a formal specification language is "Z" (pronounced "zed" in British style), which was developed at Oxford University and is now gaining momentum in Britain. Z is well suited to functional specifications. Other kinds of specification languages are targeted primarily toward more specific problem domains such as display-screen description, processor performance, reliability, and the like. The capability being specified may be of a type for which an executable program performing its functions is the most useful form of specification, as will be discussed later.

Prototyping is a technique for gathering information by building and executing a piece of software designed to answer questions about how the final product should work. In a system context, prototype software may assume other roles, such as that of simulating a portion of the system that is not yet finished.

Software prototyping is useful where complexity exceeds our ability to grasp a written design specification or requirement (e.g., estimating on-board signal processing throughput for new aircraft and sensor combinations, defining software architecture for intelligence fusion tasks using parallelism or rule-based methods, or most of the complex human-machine interfaces). These are difficult for users to properly conceptualize without observing prototype operation.

Executable Specifications. Executability and specification can be combined into single languages. VHLLs are very high level (i.e., less detail is required). They are specifications (i.e., they can express the same function as an implementation in an HLL, such as Ada). They are

formal (i.e., they have agreed upon, well-defined, communicable meaning). Also, programs written in them can be executed to provide behavioral information. It is appropriate to consider compilable VHLLs as languages suitable for preparation of executable specifications. This does not imply that other languages cannot be used to write such programs, merely that it should be far easier, using a VHLL, to explore the design space available to the application.

Software Process Models. Software process models describe the activities, products, organizations, personnel, and tools involved in the definition, development, deployment, and maintenance of software systems. In particular, process models focus on the *order* in which software definition and development activities are pursued, and the *transition criteria* for progressing from one set of activities to the next. The formalization of software process models presents an opportunity for increased flexibility in software acquisition, development, and maintenance. It is necessary to understand the lower-level languages needed for composing process models that adequately reflect the nature and needs of the customer, developer, maintainer, and user organizations; the target system architecture; the prevailing software technology; and the availability of resources. It should be noted that for unprecedented systems, the process model may not be static but may well evolve as part of the process.

The representation and support of formal process models in automated software management tools promises to increase software project transparency to both customer and developer. Most important, such automated tools encourage the exploration of alternative plans and their evaluation with respect to the needs of participating organizations, risks involved, and resources needed. Improved analysis in planning and greater flexibility in replanning will pay off in better adapted, more efficient processes and will result in better products.

Technology Insertion. The proper point in a program for technology insertion is a critical issue. The most evident application is in the early phases of the software life cycle, as requirements are converging to a better-defined specification and prototypes are being built to gain design or requirements information. This phase should not be encumbered with nonessential regulations and required use of low technology; newer methods must be acceptable here.

New languages and tools will evolve incrementally, with high priority capabilities appearing first. As an example, an object-oriented data base or fourth-generation language now used for prototyping a management information system may not yet have real time capabilities or sparkling execution performance, but it is still the most effective approach for modeling data manipulation. As another example, can a particular network control concept for some large project be scaled up to the eventual number of terminals that must be supported? A prototyping tool may address only the issue of large networks and not yet have a full range of capability. Still, it may well be the only reasonable way to build a prototype to determine if the concept can be sized to a different system.

Communications and Data Organization for Software Development

Technology can help close the communication gaps among designers, programmers, integrators, and project managers as well as among personnel in parallel roles, such as programmers coding interfacing modules. Current practices aimed at filling these gaps consist of natural language. Daily, weekly, or monthly group meetings are

often unstructured and informal, and thus cannot overcome inconsistency and ambiguity. If these meetings are structured and formal, they are usually untimely for handling real time communication gaps. When the size of software teams grows beyond a dozen individuals such techniques become even less effective.

Technology can go much farther in reducing these gaps by having the SEE provide a single central clearinghouse for information to be used by people in all the different project roles. The SEE should also provide a set of tools for using the information to support each role. Such information may be simple data, relational data, object-oriented data, and knowledge-based data. As information is organized, its utility, integrity, and value increases. Each type of data organization is characteristic of certain aspects of software development, and each is manipulated in characteristic ways.

Simple data may consist of the actual source code for a specific version or configuration of a software module(s), a set of software trouble reports, the results of a test run, and the estimated level of completion for each module. Tools to manipulate simple data include cost models, text editors, test coverage analyzers, and simple configuration management tools.

Relational data can include all of the above, as well as tell which data are related and should be considered together when changes or additions are made. For example, relationships between modules, subsystems, or systems can be stated such that a change in one with regard to units or precision can flag the need to check the changes in the others for possible impact on their software. Tools working with these kinds of data can automatically identify which modules may be affected by a change, the impact of a change on the development and support life cycles, or even the estimated impact of personnel loss, and hence can provide more powerful communication mechanisms than those working from simple data.

Object-oriented data are the result of treating the software development process as the creation and management of objects relevant to the problem's solution. Information is supplied on what the object can do, with what it can interface, and how the interfacing is done. An example could be an object-oriented digital mapping system where objects such as cities are defined as connected by roads, rail routes, airports, and sea ports; objects such as trucks can move personnel and material on roads; and airplanes must use airports, etc. Supporting tools can make much more sophisticated analyses to insure that objects developed or manipulated by one piece of software are consistent and compatible with others, and thus provide more "intelligent" communications between roles and team members.

Knowledge-based data can include all of the above, as well as generic data such as rules to determine (1) what constitutes a need to recalculate the cost to complete, (2) how to perform the recalculation when a task is getting behind, (3) which algorithm is most appropriate for a specific goal solution, and the like. Relationships within classes of objects such as vehicles, planes, or even subsystems can be easily expressed. Knowledge-based data can also include some domain-specific knowledge such as the degree of specification to be expected from a specific radar, the amount of time in which a task must be performed, and the bandwidth of a communication system with which the new system must interface. Software engineering tools using these data can perform tasks normally attributed to "experts," such as true automated configuration management, project management, requirements analysis, and

performance enhancement. Communication in this plane is much more efficient and powerful since tools can screen out nonessential data (based on role or person), automatically produce as well as trigger needed interaction, and actually create the communication.

Today, the technology to support communication using simple, relational, and object-oriented languages is well in hand. Environments that supply the underlying data bases and user interfaces, as well as a framework for adding new tools to operate on these data, are being demonstrated and marketed. Examples are the Rational System, the Rome Air Development Center (RADC) Software Life Cycle Support Environment (SLCSE), and a host of others that focus on subordinate parts of the approach, such as object-oriented design.

Knowledge-based data have an active research program under way, funded largely through the Air Force's Computation Sciences Program at RADC, with both sponsored and unsponsored participation from industry, academia, and the United Kingdom. Demonstrations of the approach in support of project management, requirements analysis, specification, and performance enhancement already exist.

At the knowledge-based systems level of abstraction, the traditional separation of program elements and data elements is no longer complete. At least some of the operational information (such as rules and formulas) is manipulated freely during execution, unlike traditional programming concepts in which variable data is moved past a static program. Since conventional processor designs embody many of the concepts and restrictions imposed by the traditional view of data as composed only of simple data, some time may elapse before systems are able to perform knowledge-based operations with field-acceptable performance. The power of knowledge-based systems is already proving attractive for many non-real-time applications, and will eventually stimulate new hardware/software architectures.

3.6.3 Test Technology

The Air Force should take firm steps to advance the automated software test technology base. This area has not been emphasized, but rather has been mostly ignored throughout the development and evaluation of Air Force systems. This situation is closely analogous to the popular oil filter advertisement to "pay me now or pay me later." Advanced technology will require more stringent discipline for testing software before completion of development, test, and evaluation (DT&E). It will also enhance configuration management after deployment. Advanced software test technology must be incorporated in the form of program functions, strategies, and tools when developing systems nearing or exceeding one million lines of code. Several base technologies could significantly increase the confidence level in the delivered systems. Associated with each base technology is a technique and associated support software. These technologies also would be passed on to the maintainer, within the development SEE, to permit him to maintain high software reliability throughout system life. The basis for software test technology can be found in the areas of automated test data generation, symbolic execution, test coverage analysis, and program verification. Some commercial off-the-shelf software test tools are available for use in software development. Their capabilities need to be understood by acquisition management personnel, so that solicitations can specify more stringent but known-practicable criteria for software testing.

Research efforts should be directed toward each of the test technology areas mentioned. Since no single area is expected to produce a panacea for software reliability, a mix of test technologies would be applied over the software life cycle. An example of early life cycle use would be automated test data generation from executable specifications.

RECOMMENDATION 21: AFSC should require the use of commercial off-the-shelf software test technology in system and software development, make it a part of the technology and software process R&D programs to further advance the area, and apply it throughout the software life cycle.

3.6.4 Software Quality

In most software acquisition, AFSC historically incentivized only cost and schedule, if indeed there could be said to be any specific incentivizing of software. Few formal demands have been made on the contractor with direct regard to the quality of his software. The current document-centered acquisition strategy (DoD-STD 2167A) requires the contractor to produce a test plan, get it approved by the Air Force, and then adhere to it. Only rarely is the plan scrutinized for its adequacy. How a test plan really proves its adequac is unanswered. The state of knowledge in software quality supports much more vigorous demonstrations of quality, which *can* be used as a basis for incentivizing contractors. No software quality indicators are addressed in AFSC Pamphlet 800-14, although this pamphlet seems heavily focused on latent errors in software.

Software quality should properly include all the "ilities" important for a specific system; these will vary from system to system. Examples are reliability (the latent error density in the software), maintainability (the ease of modifying or evolving the software at a later date), portability (the ease of moving the software from one system to another), re-usability (the degree to which the software provides general rather than specific solutions), and performance. Most of these are meaningful only from the system perspective, since software can help hardware recover from faults, and poor hardware can render most any software unacceptable.

These characteristics should be specified up front in the system acquisition, *and* they should be measureable at all phases of the life cycle (e.g., after modifications), so that both the Air Force and the contractors know whether they are being met. While IV&V contractors have evaluated systems quality along such lines, if quality specifications and incentives are not part of the contract, there is nothing to use to hold the contractor's feet to the fire.

Of major interest in every Air Force software acquisition is software reliability. Tools already in the marketplace can help the service get more mileage for each testing dollar. Elsewhere in this report we have advocated using techniques and tools to reduce major sources of critical and expensive software errors in requirements analysis and to translate requirements into a working program. However, other tools exist or are in development that can help eliminate errors and provide a higher degree of confidence in the software.

For example, tools exist to measure the effectiveness of testing, by identifying those pieces of the software that have been unexercised with the test cases. Some of these tools or combination of tools and techniques (e.g., test coverage analysis and symbolic evaluation) can help generate test cases to

drive software down specific paths. As in other areas of defense work, many Air Force contractors treat software as a labor-intensive assembly-like operation, eschewing modern tools because of their capital cost. Many contractors believe that labor-intensive defense work is best because capital investments cannot always be recovered. Tools related to software quality can and should be part of SEEs, mandated through action pursuant to our Recommendation 8.

Another way to measure testing effectiveness is to seed the software with known errors to determine if the test cases can detect them. This approach has been automated through a technique known as Testing Mutations, whereby a tool alters the source program to produce a "mutant," and submits the program for execution to ascertain if the test cases detect the mutant. Further experimentation has combined this technique with symbolic evaluation to produce computer generated test cases that can "kill" the mutants. This technique has been used in parallel processing where different mutants are generated and tested in parallel, thereby cutting the cost and time of such an approach. Of course, proof that the mutants act the same as actual design or implementation errors cannot be provided.

Yet another approach uses a piece of the Quality Measurement Framework developed jointly by RADC; Army Institute for Research in Management, Information, Communications, and Computer Science; and Defense Mapping Agency to specify, predict, and measure software quality. The piece of this technology that focuses on software reliability and which analyzes the software for characteristics supportive and not supportive of reliability (such as complexity, localization of data, and modularity) has been demonstrated to provide analyses that directly correlate to the residual errors in a delivered system.

This same Quality Measurement Framework can analyze software for characteristics supportive of other "ilities" such as maintainability or enhancability for pre-planned product improvement (P^3I), re-usability, efficiency, integrity, and other qualities. The framework provides a measure between 0.0 and 1.0. Currently, this Framework is used on the Army worldwide information system (WIS) Program, and by several IV&V contractors, and has been modified by at least one Japanese company for internal use.

The above technologies and others (e.g., formal verification or critical error analysis) can be used to improve software quality. The Air Force can now demand demonstrations of quality, for example, that every instruction be executed at least once during testing; that the overall reliability, maintainability, re-usability, etc. be within a specified range (e.g., .95 to 1.0); or that a formal analysis be undertaken to show the absence of certain critical errors (e.g., unwanted release of a weapon or activation of an EW system).

3.6.5 Integrated Tools and Techniques for Better Integrated System and Software Development

We believe that the current approach of separately managing and developing software and hardware (including sensors, engines, weapons, radios, etc., as well as computing hardware) is inappropriate for modern system development. Software and hardware are becoming more and more intertwined. The traditional separation can only adversely affect the cost, schedule, and performance of the system.

Almost every weapon subsystem, avionics system, sensor, control mechanism, power unit, and life support system aboard an aircraft contains software. To fulfill its intended purpose, this

software must often communicate with the software, hardware, and performance of other systems and subsystems to which it relates. Forced separation of hardware and software, and of the design approaches to each, impedes the necessary integration and cooperation of the hardware and software.

The same tools and techniques used and proposed to develop and manage software can be extended to support development and management of *the system itself*. For example, the tools for prototyping do not and should not distinguish between hardware and software functions or performance. The tools that formally record requirements or specifications do not distinguish between hardware and software.

There is also no need to separate the two entities at later phases in the life cycle, even after they take on separate forms. Throughout computer history, changes in expected performance of the hardware have had direct impact on the software, whether the changes were positive or negative; the converse is also true. Schedule slips in elements of hardware or software can put elements of both on the critical path, and the use of separate management tools for systems and software only delays the bad news. Changes in configurations of hardware or software often influence each other, or should.

Software traditionally has had to deal with complexity and volatility of requirements, specifications, designs, personnel, and schedules to a greater degree than has hardware. The software field has formal tools to deal with these problems to varying degrees. In this respect it is at least slightly ahead of systems technology, where the tools are less formal. Computer Aided Systems Engineering (CASE) is a recent arrival, thus far lacking the coherence of existing software tools and techniques. If we are to handle the increasing interdependencies of hardware and software from a total systems perspective, it makes sense now to build from the tools and techniques of the software world, since software is leading in complexity, functionality, and performance issues.

As suggested earlier, tools for requirements analysis, prototyping, and specifications (the front end of the life cycle) already do not distinguish between hardware and software. At later stages, tools needed by hardware and software teams become distinct, but in both cases are becoming more automated. Both have begun to create data bases (in some cases knowledge bases) that can be integrated with some serious effort. This integration can be at several levels: simple data, relational data, object-oriented data, and knowledge-based data, with the potential synergy increasing as one moves from simple data to the knowledge-based paradigm. These data can then be integrated across the life cycle in one life cycle engineering support environment.

Ideally, one might want to start from scratch to design a totally integrated systems engineering environment. In fact, Project Forecast II recommended, through a technology program known as United Life Cycle Design, that a single environment or set of environments be put together to allow all aspects of a system to be treated together, from conception through fielding and support. Extending the purpose and in some cases the functionality of the software tools and environments and integrating them with the hardware life cycle tools will let us gain the early experience necessary to undertake such a task and provide near-term products. Since many functions that will be needed in system and software engineering and management are still poorly understood, it is inappropriate to suggest development of a fully operational capability. Rather, a prototype should

be initiated, followed by incremental evolution of the eventual integrated system/software engineering and management tool.

Quantitative Evaluation of Software Quality

Software quality may be understood from two perspectives, the products of the process (requirements specifications, designs, source code, error reports, etc.) and the process itself (organization, methodologies used, the Q/A approach, etc.). Today technology and methods to support the former are much farther along than for the latter, and are ready to be implemented at least on a trial basis, in actual system developments.

Quality product evaluation consists of assessing reliability, maintainability, re-usability, and the like by focusing on metrics that can be evaluated automatically by a tool or by simple "yes or no" questions put to a human. A software product quality strategy, framework, and a set of automated tools known as the Automated Measurement System, are already at RADC in prototype form and used by at least one service (Army WIS program) and several IV&V contractors. This prototype can help a program office specify the degree to which the system will achieve the "ility," predict how well the product will comply by evaluating the interim products, and then provide a final assessment. This prototype has been hindered in its calibration of the metrics by a shortage of data on real systems but can be of substantial help if this shortage can be overcome. If handled properly, this approach, though innovative, should not cause performance or schedule problems; however, 3 to 5 percent of the software budget may be needed to collect and analyze the data.

RECOMMENDATION 22: AFSC should select key programs that have high concerns for reliability, maintainability, re-usability, and interoperability for demonstration and evaluation of this prototype product quality assessment scheme. AFSC should invest funds to merge product and process quality measurement schemes to get increased benefits and to keep the measurement technology updated to the needs of future life cycle models.

Process metrics research is under way at several places. By far the leader in this area on the scales that need to be addressed for large systems acquisitions is the Software Engineering Institute (SEI). SEI's program is aimed at providing assessments of contractor's software development processes. Other significant work is going on at MITRE, where process metrics are used to help in pre-award assessments of contractor capabilities.

3.6.6 Software Technology Transfer Activities

DoD has treated software technology activities entirely as program element 6.X (research, development, test, and evaluation) actitivities. In this report, our emphasis is not principally on software technology, but on the *application* of that technology to system and software development. It is time that software advanced beyond an image of combining arcane technology and cottage-industry production. Formal steps can and should be taken to transfer technology into specific programs needing it. For most Air Force technology, "tech transfer" occurs almost by osmosis as contractors move a technology through 6.2 and 6.3 programs to application in a 6.4 FSED system program. This route is most appropriate for technologies that will be embedded in systems: component technologies and materials technologies. Design and manufacturing technologies are not embedded in systems and have no corresponding path (i.e., it makes little

sense to first apply a computer design technology to the design of advanced prototype computers). Rather, design and manufacturing technologies must be used to design or manufacture systems that are reasonably well understood. MANTECH (manufacturing technology) programs in the Air Force and other services are examples. The Defense Science Board, in a 1988 software workshop that followed the report of the DSB software panel chaired in 1985 by Dr. Fred Brooks, suggested a software-based program corresponding to MANTECH. We believe this is an important idea: software technology differs in many respects different from other important technologies, and calls for different methods of technology transfer.

We leave to AFSC the interesting challenge of naming such a program (the term SOFTECH is otherwise in use, but CASETECH is possible). However, in line with MANTECH, the activities of this program should be carried out by the contractor people who will develop next-generation systems, rather than by disconnected research or technology groups having little sense of the problems in a large software development effort. Some features may be like those of MANTECH, for example, having each project related closely to the needs of a real system whose development is about to get under way or which will begin in the near future. The program should acknowledge the need for substantial capital investment--both software and hardware--though this should presumably be only a portion of what will be used in the actual large-scale software development to follow.

RECOMMENDATION 23: AFSC should initiate a program in the style of MANTECH (manufacturing technology), to transfer software development process technologies into actual major system and software development programs.

This program could also allow for the necessary transformation of technologies from the laboratories, industry, and academia into "industrial-grade" tools. Should AFSC decide it needs an SEE or a set of SEEs, the program could fund the evolution of this environment by properly engineering the tech-base products from various sources. The program should be managed by an AFSC organization having both software engineering technology background and major involvement in systems acquisitions. It could be distributed between the product divisions involved in software acquisition, and their associated laboratories, but it is essential that it be tasked to deal with AFSC-wide as well as system-specific needs.

Software Technology Demonstration Projects

Traditionally, AFSC SPOs have understandably backed away from software approaches that could be perceived as adding to the cost, risk, or overall responsibility of the government. As a result, software technology has been introduced slowly because it has been unable to mature or prove itself in system acquisition.

Funds are needed to cover additional expenses incurred with the investment in workstations or tool sets; the application of software quality metrics, more vigorous testing strategies, automated communications between buyer, user, and development teams; and the reporting of productivity and quality data associated with the above. If the "culture shock" is suitably high, such as in the introduction of VHLL for prototyping, the funds could go to a shadow program so as not to influence the risk or schedule.

These projects, analogous to STARS' "Shadow" projects, should be included as a function of the software

process technology transfer program.

3.6.7 Software Technology Programs

Software technology can make or break U.S. national security. Its very ubiquitousness has apparently denied it the attention and resources given more narrowly employed technologies such as stealth, composite structures, or solid-fueled rocket propulsion. Many people are so blinded by the impressive software products of the U.S. computer industry and the ubiquity of PC software that they do not realize that most Air Force software needs cannot be anticipated by, nor extrapolated from, these limited but visually impressive products.

As an example of the shortfall in software technology, with the exception of DARPA's Strategic Computing Program, we are not aware of any significant Air Force software engineering technology activities directed toward parallel, distributed, and non-Von Neumann architectures; yet only these architectures can provide the performance or survivability needed for future systems.

A similar case can be made for the prototyping-VHLL, software quality, and other technology programs aimed at providing rapid requirements conversion and Dem/Val for systems software. Outside the Air Force technology base software engineering program, which is quite modest, little effort is going into such programs. It is unreasonable to expect the commercial software industry to provide for Air Force's needs in these areas; DoD is virtually the only customer that requires enormous software and system ensembles built to meet its specific needs.

What is the "right" level of overall software technology support? For most of the critical technologies used by AFSC, the level is established by the following prescription, which can be applied during the planning, programming, and budgeting system (PPBS) planning cycle:

1. The technology program should include several competing pursuits for each critical technology, to increase the chances of success and avoid complacency by researchers.

2. The technology program should include coherent attack of all recognized opportunity areas and all unavoidable problem areas pertinent to Air Force systems.

3. The technology program should address technologies needed by projected new or modified systems at a pace that can be expected to produce results in time for system application.

At present, software technology programs do not meet these criteria because software has been treated as nonessential technology.

The "right" software technology program must be the result of a broad and studied analysis of these three criteria by Air Force management. This analysis should be carried out as part of the normal PPBS process, although it is currently most appropriate that each product division and each relevant laboratory (including RADC, AFWAL, Human Resources Lab, Aerospace Medical Research Lab, and Space Technology Center) address software opportunities and needs. RADC, the technology organization with the greatest experience in this area, should be given a special and perhaps coordinating role, but the other parts of the technology organization must become involved in the problems and opportunities. Since software is ubiquitous in Air Force systems, it is reasonable that many laboratories be involved, each doing what is best suited to its mission.

The software technology community believes that the Air Force's software technology base is unpredictable. Because software is such a critical area and because the people and time costs of turning technical projects on and off is even greater than in mundane efforts, we recommend that continuous support for technology base support teams across contract phases be provided, and that sole source contracting be acknowledged as an acceptable tailoring option for early phases of 6.1, 6.2, or 6.3A projects. During these phases only one contractor has the desired capabilities or proprietary software resources.

RECOMMENDATION 24: AFSC should increase its technology base investment in software engineering technology, which is currently running at less than $8 million per year. This increase should involve Air Force laboratories more broadly and directly than in the past and include AFSC-wide coordination. The rate of increase of the technology program should be significant, but must be based on the capacity of the technology resources in defense industry and research institutions with coupling to systems developers to grow gracefully. As a way to improve software technology transfer, and in line with its usual strategy, AFSC should select system programs for application and demonstration of advances in software engineering technology, and provide separate 6.3 funding to support demonstrations.

Activities Supporting Software Engineering Environment Evolution

There is wide interest in the use of SEEs from industry, and in this report we have recommended that AFSC specify a minimum proven software environment for each system acquisition. Such an environment could help the Air Force keep in step with advancements in technology, and could help make the total software life cycle approach more productive by:

- supporting some variety of software development methods or strategies, perhaps through modularity;
- supporting multiple high-level languages so that the old software in JOVIAL, FORTRAN, COBOL, etc., can be supported or re-used with Ada;
- allowing easy reuse of familiar tools and integration of new tools; and
- allowing simultaneous development and support of multiple systems.

Several environments are candidates in this area, although for the most part these are very challenging objectives at present.

RECOMMENDATION 25: The AFSC should consider funding a continuing program to evaluate candidate SEEs and where applicable stand-alone tools, against the five criteria listed above, for consideration as acceptable environments and tool sets.

This recommendation directly enhanced Recommendation 8 for future normalization and improvement of SEEs used by contractors and by AFSC.

Advanced Support for the Software Systems Engineering Advisory Team(s)

We have recommended that a "nest of owls" be used to advise on some aspects of software acquisition, such as risk assessment, tailoring of user involvement, tailoring of the DoD-STD-2167A requirements, and contract form. Knowledge-based systems technology could collect and preserve the knowledge and approaches used by the members of the team(s) to provide this advice. This

practice may help with the problems of rapid turnover and inexperience of personnel involved with the system acquisition, and possibly even increase the effectiveness of the "nest of owls." A later use would be as training facilities through a "software wargaming" program. These resources would complement current work on knowledge-based software assistance that RADC is supporting.

RECOMMENDATION 26: AFSC should create and fund a project to provide support for the software systems engineering advisory team(s) of Recommendation 8, in particular to capture the knowledge gained and used by the team members for use via knowledge-based tools. This could be a valuable lead project for later use of similar tools, more broadly in AFSC system and software acquisition management.

CONCLUSIONS AND RECOMMENDATIONS

4.1 THE ACQUISITION PROCESS

We recommend appropriate application of alternative processes including adding a demonstration/validation (Dem/Val) phase, or competitive first-phase procurement, along with maintenance of a warm industrial base by funding application-specific software technology programs.

4.2 ACQUISITION POLICY

We recommend that the Air Force Systems Command (AFSC) issue a policy statement and implement workshops in tailored acquisition with special emphasis on software. The handbooks supporting Standards 2167A and 2168 should be expedited as parts of this initiative, which should also address user and maintainer involvement. Current Air Force regulations suggest use of the software maintainer (e.g., Air Force Logistics Command or AFLC) for Independent Validation and Verification (IV&V); this has worked well in the cases studied, and merits wider adoption. AFSC must avoid firm fixed price contracting in acquisition of unprecedented systems, without adequate prior risk reduction efforts.

We believe that the time is right for AFSC to join in this activity. Also:

1. Contractors should be required to use a software engineering environment (SEE) appropriate to the acquisition's needs for all but the smallest softwaredevelopment acquisitions.

2. Developers of unprecedented systems should be required to prove their possession and knowledge of suitable design exploration tools.

3. AFSC should acquire those elements of an SEE that will support computer-based documentation and build up a management tool set. Contractors should be required to deliver software engineering data in machine-readable form that can be manipulated by AFSC's document-preparation tool(s).

4.3 IN-HOUSE EXPERTISE

We recommend that experts now in the Systems Command be identified and that some portion of them be organized into one or more software systems engineering advisory teams based on knowledge and experience. These individuals should be centrally assignable and used at key stages before, during, and after contract awards involving significant software components. Additional efforts should be mounted to identify and develop system engineers with strong software skills, and software engineers with system competence. Such individuals are particularly important in AFSC, although they are needed also in industry. The scarce resource identified or made part of the advisory teams should be intensively managed through tailoring of career progression and special monitoring. Military software specialists should perceive a practicable path to grade O-6 with evidence that they could advance even further.

4.4 TECHNOLOGY PROGRAM

The needs for process-directed software technology have parallels to

those for MANTECH (the manufacturing technology program). Accordingly, we urge the Air Force to establish a *production-funded* program similar to MANTECH but for software production technology pertinent to particular systems or system classes. Commercialization of software process technologies should be encouraged, as it will improve available tools, at less cost to the government. We suggest that the software technology base be expanded at the rate it can sensibly absorb, over a period of five years or more, to bring it into closer parity with other critical technology areas.

4.5 RECOMMENDATIONS IN THIS REPORT

RECOMMENDATION 1: AFSC, working with the Joint Logistics Commanders organization, should ensure that development models and accompanying rational alternatives to the waterfall model, based on risk reduction concepts, are included in the forthcoming Handbook 287 for DoD-STD-2167A, with supporting direction in AFR 800-2 and 800-14.

RECOMMENDATION 2: AFSC should ensure adequate software risk reduction for unprecedented systems during a Dem/Val or equivalent phase preceding full-scale development. For unprecedented systems, AFSC should provide policy guidance for competitive two-phase procurements, such that software risks are reduced to a practical minimum before proposals are prepared for the following phase(s). For unprecedented systems, AFSC should direct an independent assessment of system and software risks near the end of the Dem/Val phase or Phase 1 of a two-phase procurement, as part of the basis for follow-on procurement decisions. For multiphased procurements, AFSC should direct that contract mechanisms provide continuity of essential contractor skill and expertise between contract phases.

RECOMMENDATION 3: For key technologies in systems and application areas where operational threats or requirements change rapidly, AFSC should fund parallel technology programs at the system level to foster a ready industrial base from which to compete single phase system acquisitions.

RECOMMENDATION 4: Each program involving software should be required to carry out early identification of critical software issues and to develop and maintain a Software Risk Management Plan. This instruction should be included in the most appropriate Air Force management directives.

RECOMMENDATION 5: When a program manager is faced with late identification of software requirements that can be deferred to a later time or capability block, AFSC management guidance should encourage and support this deferral and accept the consequences of doing so.

RECOMMENDATION 6: User involvement should be tailored for each program, varying from cases requiring very limited involvement to ones in which a user will assume the lead role.

RECOMMENDATION 7: AFSC should direct its product divisions to tailor the contract form for each specific program's needs; in particular, AFSC should avoid using firm fixed price contracts for unprecedented programs. (This will require management followup, consistency, and the support of higher authorities.)

RECOMMENDATION 8: Product divisions should be directed to specify use of an SEE for each program having, as an example, a software staff of more than 12 people, and to require proof of its existence and the contractor's knowledge of its effective use, in order to qualify.

RECOMMENDATION 9: When software development contracts are granted to design groups that are organizationally or geographically separated, near-term management criteria in source selection should emphasize use of modern telecommunications and division of tasks to reduce requirements for interface among separate locations or organizations. In the long term, contractors should demonstrate availability and use of a software management and engineering environment designed for such dispersed efforts.

RECOMMENDATION 10: Product divisions or Headquarters AFSC should regularly monitor computer resources working group performance. Explicit evaluations should be solicited from using commands and AFLC.

RECOMMENDATION 11: AFSC, with AFLC and the using commands, should sponsor a fresh look at actual maintainer documentation needs. This review should consider the growing automation of documentation by contractors, and how that might be be used to reduce the cost or improve the utility of the data.

RECOMMENDATION 12: AFSC should select an appropriate program (or programs) through which to implement incremental acquisition, using it (or them) to articulate to the Office of Management and Budget and the Congress the need for and special benefits of an evolutionary, incremental, acquisition approach.

RECOMMENDATION 13: AFSC must strongly encourage AFLC and the using commands toward collocated support for software in integrated systems, rather than complex reprogramming without adequate resources in the field.

RECOMMENDATION 14: AFSC should broaden the base of its personnel skilled in acquisition of software-intensive systems; prepare, use, and maintain current guidebooks; and exercise special management of the skilled personnel resource.

RECOMMENDATION 15: AFSC special management of software skills should include a software systems engineering advisory team (a "nest of owls") and special career tailoring for selected officers and civilians.

RECOMMENDATION 16: AFSC should take steps to increase the motivation for innovative acquisition tailoring. AFSC should issue a policy statement, conduct workshops, and distribute guidebooks.

RECOMMENDATION 17: AFSC, in collaboration with others, should make available to officers and civilians a mid-career systems engineering and software engineering graduate program and appropriate short courses.

RECOMMENDATION 18: The Air Force should consider revision of AFR 800-14, paragraph 5-3, Test Planning, and all derivative directives, to require demonstration of testing of every instruction within the software prior to completion of development, test, and evaluation (DT&E). Implementation needs, cost, and expected benefits should be analyzed by experts prior to implementing this revision.

RECOMMENDATION 19: Each program should consider using the designated software "maintainer" (operational phase) as the independent validation and verification agent during software development.

RECOMMENDATION 20: AFSC, with the Joint Logistics Commanders, should expedite preparation and distribution of the 2168 guidebook and support maintenance of this and other software guidebooks over time.

RECOMMENDATION 21: AFSC should require the use of commercial off-the-

shelf software test technology in system and software development, make it a part of the technology and software process research and development programs to further advance the area, and apply it throughout the software life cycle.

RECOMMENDATION 22: AFSC should select key programs that have high concerns for reliability, maintainability, re-usability, and interoperability for demonstration and evaluation of this prototype product quality assessment scheme. AFSC should invest funds to merge product and process quality measurement schemes to get increased benefits and to keep the measurement technology updated to the needs of future life cycle models.

RECOMMENDATION 23: AFSC should initiate a program in the style of MANTECH (the manufacturing technology program) to transfer software development process technologies into actual [illegible]or system and software developm.. programs.

RECOMMENDATION 24: AFSC should increase its technology base investment in software engineering technology, which is currently running at less than $8 million per year. This increase should involve Air Force laboratories more broadly and directly than in the past and include AFSC-wide coordination. The rate of increase of the technology program should be significant, but must be based on the capacity of the technology resources in defense industry and research institutions with coupling to systems developers to grow gracefully. As a way to improve software technology transfer, and in line with its usual strategy, AFSC should select system programs for application and demonstration of advances in software engineering technology, and provide separate 6.3 funding to support demonstrations.

RECOMMENDATION 25: The AFSC should consider funding a program to evaluate candidate SEEs and where applicable, stand-alone tools, for consideration as acceptable environments and tool sets.

RECOMMENDATION 26: AFSC should create and fund a project to provide support for the software systems engineering advisory team(s) of Recommendation 8, in particular to capture the knowledge gained and used by the team members for use via knowledge-based tools. This could be a valuable lead project for later use of similar tools, more broadly in AFSC system and software acquisition management.

APPENDIX A

PREVIOUS SOFTWARE STUDIES WITH BRIEF SUMMARY OF RECOMMENDATIONS

1. *A Study of Critical Factors in Management Information Systems for USAF*, prepared for Air Force Office of Scientific Research by Carter, Gibson and Rademacher, Colorado School of Mines, March 31, 1975.

A study to address the development of management information systems in the Air Force, identify critical factors for successful development, suggestions for measuring such factors and embodying them in a predictive model.

2. "Software," a special issue of *Defense Management Journal* devoted to DoD software matters, October, 1975.

Jacques Gansler opens with editorial comments. Fourteen articles covering overview, management factors, embedded systems, configuration management, personnel quality, and metrics policy.

3. *Operational Software Management and Development for the USAF Computer Systems*, Air Force Studies Board, summer study, 1977. Gordon Heffron Panel.

Interacted with a broad variety of military (B-1, 485L, AFR 800-14, National Software Works, Shuttle Software, Safeguard) and commercial software efforts.

A. Development contract should be preceded by a separate contract for detailed systems design requirements (possibly on a competitive fly-off basis).

B. Development should proceed through a series of deliveries, each standing by itself and not requiring rework of prior ones [incremental delivery].

C. Obligate the system program office (SPO) to consider life cycle support during his development effort.

D. Advance the state of software tools.

E. Extend period of assignment of officers to the SPO to exploit limited expertise; develop short courses for software personnel.

F. Maintain a vigorous program to improve the technology base.

G. AFSC should strengthen its support of Air Force in software matters, and also provide a focal point for software (in XRF).

4. "Software Management in the Air Force," Air Force Scientific Advisory Board, 16 June 1978. [Dave Walden, Mike Muntner, Ike Nehama, MG Gerry Hendricks, Capt Danny Beam]

   A. Continue to use modern software development technology.

   B. Identify software as a potential high technology risk area (not merely an area of cost risk).

   C. Modify procurement practices (or bend existing ones to the limit) to the realities of the software development process.

   D. Freeze the methodology (or at least do not encourage development of new methodology).

   E. Initiate data collections [to get software metrics and cost estimation].

   F. Stop expecting software alone to be the point of all system integration.

   G. Use V&V at an early stage.

5. "Software Requirements for Embedded Computers," Glaseman and Davis, Rand Corporation, March 1980.

A discussion based on examination of eight major programs at Electronic Systems Division and Aeronautical Systems Division. No recommendations given but notes problems of lack of information to guide the design and implementation and costing of software efforts, personnel shortage, lack of a structured approach within Air Force to software projects.

6. *Government-Wide Guidelines and Manage Assistance Center Needed to Improve ADP Systems Development*, General Accounting Office, February 20, 1981.

Refers to "over 57 reports" from GAO in 10 years noting weak management in development and procurement of large computer systems within government. Calls for Office of Management and Budget guidelines for a structured management approach.

7. "Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged," GAO, February 26, 1981.

Based on 15 site visits and hundreds of questionnaires, summarizes the sorry state of software life cycle support and its management throughout government.

8. *Survivable Tactical and Strategic C^3I Systems*, George Mueller Panel, AFSB, March 1981. Classified report.

Concentrates only on C3I matters and their survivability and details. Software is not mentioned; no recommendations on software are given.

9. *Defense Automatic Data Processing Acquisition*, Headquarters, United States Air Force, by Booz Allen, 30 November 1981; the Sawyer Study.

   A. Make sure that ADP program is mission-oriented (mission primacy).

   B. The acquisition process must be changed to provide flexibility to accommodate changing needs and environment. [During development and during LCS]

   C. Internal [Air Force] roles, responsibilities, and missions must be brought into line with the change of [B].

   D. Outside the Air Force policy and organizational modification must take place to complement Air Force changes and to support them.

10. *Report of the DoD Joint Service Task Force on Software Problems*, Larry Druffel Panel, Defense Science Board, July 30, 1982.

   A. Create a DoD-level software initiative for embedded systems with strong cooperation to Ada and VHSIC initiatives.

   B. For [A], create a plan of action, and each Service continue its efforts to address software issues.

   C. Re [A], initiative should address acquisition/management (LCS, contract monitoring, contract incentives, microprocessor and firmware policies); technology (system design, impact of requirements on systems, data collection and standards for metrics, common tools, documentation support, strategy for handling rapid technology change, cross-service sharing and research and development (R&D) communication, technology transfer and infusion, software R&D within IR&D), and expertise (increase skills and population of management and software people, government/industry exchange, productivity measures for software implementers).

Appendix B summarizes 10 Navy studies, 4 Army studies, and 12 Air Force studies with over matrix chart showing areas discussed in each, and a one-page summary.

11. *Multilevel Data Management Security*, Air Force Studies Board, summer study, 1983. Marvin Schaefer Group.

No recommendations on software per se. Addresses the title directly.

12. *The High Cost and Risk of Mission-Critical Software*, Air Force Scientific Advisory Board, December 1983. Jack Munson ad hoc committee.

Major General Recommendations:

   A. Establish a focused high-priority career path for software and computer system personnel.

B. Create a plan to evolve a Deputy Chief of Staff-level manager of Air Force information resources, including [both] mission-critical and embedded computers and software.

C. Establish a software policy on risk management.

D. Provide risk management information support (e.g., cost, schedule, tools).

E. Establish a policy on software oversight management (i.e., oversight managers at Division and Systems Command level to retain high level visibility of potential and actual software issues).

F. Increase the investment for software engineering and tooling.

G. Increase funding for long range software [production] technology research.

H. Acquisition practices must encourage technology transfer.

I. Increase use of independent validation and verification (IV&V).

J. Investigate methodology to ensure high integrity software.

K. Upgrade post deployment software support personnel.

L. Standardize and increase capital spending for post deployment softwre support environments.

M. Enable early participation by designated support organization.

N. Establish single management authority for assuring system integrity after deployment (at program management responsibility transfer).

NOTE: The above report was paraphrased and summarized in a 14 Jun 88 memo to LE summarizing the Air Force response to the study. Many of the points noted are the same, but interpretive comments were added. The latter have been added above either in parentheses or in square brackets.

However, a few are phrased quite differently than in the original report. The different ones are:

A. Better earlier understanding of software requirements.

B. Invest in R&D on software process dynamics; and increase the supply of software personnel.

C. Establish policy and procedures for uniform tracking and control of software development.

F. Establish a software engineering and computer system technology and support center to collect and focus the AF on software issues.

13. *Initiatives to Improve the Development of Air Force C3I Software*, MITRE Corp, March 1984, vol I., Zraket and Campbell.

    A. Create a broader overview for software technology at Electronic Systems Division (ESD) through organizational and programmatic changes.

    B. Implement mission-line development and rapid prototyping facilities.

    C. Accelerate transfer of AI technology to ESD programs.

    D. Create at ESD a facility to promote reusability of software.

Surveyed a very large number of projects and interacted widely with industry. Vol II contains detailed background of C3I software development problems.

14. *Methods for Improving Software Quality and Cost*, Air Force Studies Board, summer study, 1984. The Charles Vick group.

Recommendations are often framed in terms of near-, mid-, and far-term aspects. Emphasis of report is on software productivity and management of the process.

    A. The Air Force should engineer several first generation [software development] environments that use compatible, proven techniques and tools. Training, distribution, and maintenance schemes should be devised from these environments. Create functional specifications for the requirements of the SDE and enforce its use in all software acquisitions.

    B. Assure completeness of the tool set and support their integration across the entire life cycle of software. Concentrate on software requirements and specification phases through [illegible] languages, and analysis/validation tools. More R&D on computer-aided test case design and analysis. Exploit use of intelligent data base management systems within the environment to help manage and integrate the data needed within the environment to support the tools. Employ "distributed software engineering work state" concept to allow all managers and engineers on a software project to communicate, interact, and share as required.

    C. Research and develop the knowledge-based software assistant (KBSA) to increase the level of automation and sophistication of the software development environment.

    D. Advocate prototyping and rapid prototyping to handle the up-front requirements and specifications resolution convergence.

    E. Create Center(s) of Expertise for software acquisition and management.

    F. The Air Force should develop and use alternate acquisition strategies, e.g., prototyping.

    G. The Air Force provide for immediate development of tools for software management of costing, scheduling, communications among offices and managers

and workers on a project, quality assessment, and assurance.

H. Air Force should give better training and performance incentives for management and software personnel.

I. Air Force should impose better selection procedures for software manager.

J. Present computer architectures (including 1750-A) will be marginal for many applications in C3I and avionics.

K. A lack of tools for more elegant architectures will seriously limit Air Force exploitation of artificial intelligence, Massively Parallel Processors, direct-execution machines, etc. and therefore the Air Force will not be able to derive the full benefit of technology.

L. The Air Force should consider replacing 1750A with 1862, especially in regard to use of Ada.

M. The Air Force should do more R&D on non-traditional architectures.

N. Regarding the re-usability of software, the order of priority for attention is:

a. Reuse tools, especially within all 3 generations (near- through far-term) of SDEs.

b. Re-use models for specifications and design of applications.

c. Re-use major subsystems such as message handlers, communications protocol packages, etc.

d. Re-use Ada packages for mission applications.

In addition, the Vick group reviewed the 1976 study and generally supported and reaffirmed its recommendations.

15. *Air Force Base Level Automation Environment*, Board on Telecommunications and Computer Applications, National Research Council, January 1987. The Ivan Selin Group.

A. Identify elements of [ADP] systems that can be acquired concurrently and adjust AF procedures to permit this approach.

B. Decentralize ADP system acquisition to major commands (MAJCOMs) for all but the largest and most complex Air Force-wide systems.

C. Have the sponsoring and support commands both include funds in the POM to implement approved programs. In addition have major commands includes out-year funding for the LCS of programs.

D. Improve quality of contracting people assigned to major commands by training.

E. Modify officer assignments to have them in parallel to major commands during an acquisition or major phase thereof.

F. Expand use of requirements contracts for life cycle support, application software and technology infusion.

G. Coordinate peacetime and wartime needs better.

H. Emphasize the importance and priority of information systems at a senior level to assure proper focus and availability of resources.

I. Handle the Office of Management and Budget and Congressional interfaces better to improve state of information and awareness.

J. Make sure that field purchasing and contracting officers are well informed on Federal Acquisition Regulations.

16. *Military Software*, Defense Science Board, September, 1987. The Fred Brooks Panel. NOTE: The report was quite late. The work had been done in 1985-86.

A. Undersecretary of Defense (Acquisition) should establish a formal coordination mechanism for five relevant but disjoint initiatives. Create a coordinated DoD Software Technology Plan.

B. The STARS Joint Program Office (JPO) should be moved to Air Force ESD and a general officer should be given responsibility for STARS. Move STARS and rebuild it.

C. Task STARS director for a new set of program goals.

D. Direct STARS to choose several real programs early in development and augment their funding to ensure use of modern practices and tools.

E. Commit DoD management to a serious and determined push to Ada.

F. Move the Ada JPO into the same organization as STARS and the Software Engineering Institute (SEI).

G. Retain the Ada JPO as the technical staff support agent for the DoD's executive agent.

H. Retain DoD policy forbidding subsetting of Ada.

I. DoD should increase investment in Ada practices, education and training for both managers and technical people.

J. Focus a critical mass of software research effort on software needs unique to SDI.

K. Use evolutionary acquisition, including simulation and prototyping, to reduce risk [for SDI].

L. Undersecretary of Defense (Acquisition) should adopt a 4-category classification as the basis for acquisition policy (i.e., standard, extended, embedded, advanced). ALSO: develop acquisition policy and procedures for each category.

M. All methodological efforts (especially STARS) should see how commercially availably software tools can be selected and standardized for DoD needs.

N. DoD should devise increased productivity incentives for custom-built software contracts, and make such incentivized contracts the standard practice.

O. DoD should devise increased profit incentives on software quality.

P. DoD should develop metrics and measuring techniques for software quality and completeness, and incorporate them routinely into contracts.

Q. DoD should develop metrics to measure implementation progress.

R. DoD should follow the concepts of the proposed FAR 27.4 for data rights for military software.

S. Undersecretary of Defense (Acquisition) should update DoD 5000.29 so that it mandates iterative setting of specifications, the rapid prototyping of specified systems and incremental development.

T. DoD Std 2167 should be further revised to remove any remaining dependence on the assumptions of the "waterfall" model and to institutionalize rapid prototyping and incremental development.

U. Directive 5000.29 and STD 2167 should be revised or superseded by policy to mandate risk management and techniques in software acquisition as recommended in the 1983 USAF/SAB study.

V. Each service should provide its software Product Development Divisions with the ability to do rapid prototyping in conjunction withusers.

W. Each service should provide its software using Commands with facilities to do comprehensive operational testing and life-cycle evaluation of extensions and changes.

X. The Undersecretary of Defense (Acquisition) and the Assistant Secretary of Defense (Comptroller) should spell out by directive the role of Using Commands in the evolutionary and incremental development of software systems.

Y. The Undersecretary of Defense (Acquisition) should develop economic incentives to encourage contractors to buy modules rather than building new ones; and to allow contractors to profit from offering [software] modules for reuse, even though built with DoD funds.

Z. The Undersecretary of Defense (Acquisition) should develop economic incentives, to be incorporated into all cost-plus standard contracts and to new ones.

AA. The Undersecretary of Defense (Acquisition) and the Assistant Secretary of Defense (Comptroller) should direct program managers to identify in their programs those subsystems, components, and perhaps even modules, that may be expected to be acquired rather than built; and to reward such acquisition in the requests for proposals (RFPs).

AB. The SEI should establish a prototype module market, focused originally on Ada modules and tools for Ada, with the objectives of spinning it off when economically viable.

AC. The SEI, in consultation with Ada JPO, should establish standards of description for Ada modules to be offered through the software modules market.

AD. [Since it is unlikely that the DoD can solve its skilled personnel shortage], plan best how to live with it and ameliorate it.

AF. Establish mechanisms for tracking personnel skills and projecting personnel needs.

AG. Structure some officer careers to build a cadre of technical managers with deep technical mastery and broad operational overview.

AH. Enhance education for software personnel.

17. *Results of Follow-up 1983 SAB Report on Mission-Critical Software*, 14 June 1988, LTG Charles McDonald, DCS/L&E.

The following commands participated in the survey that this document summarizes and reports: (see item [12] above) Air Force Communications Command, Air Force Logistics Command, Air Force Systems Command, Air Force Space Command, Air Training Command, Military Airlift Command, Strategic Air Command, Tactical Air Command, and Air Force Operational Test and Evaluation Center. Each has reported its actions to the recommendations noted in [12] above. In addition, the cover memo summarizes the following seven major points as needing continuing attention.

A. Funding software R&D support resource requirements at adequate levels.

B. Including software supportability requirements in requirements documents and in the design of new or modified systems.

C. Continuing involvement by support and using/operating command personnel in the development process.

D. Sharing lessons learned and software management techniques (including standard software development and support environments and tools) across MAJCOMs and AFOTEC.

E. Supporting development of software management pamphlets and guides such as those developed by the MAJCOMs and AFOTEC.

F. Supporting MAJCOM and AFOTEC efforts to recruit, train, and retain personnel with software expertise. This includes continued support of personnel initiatives such as PALACE ACQUIRE and the Scientist and Engineer Career Enhancement Program (SECCEP).
G. Supporting the Joint Logistics Commanders' efforts to work software issues across the services.

UNDATED STUDIES

1. *Twenty Commandments of Software Acquisition*, US Army Tactical Data Systems, undated, no authors.

A series of 20 "thou shalts" apparently for the program in question. Mentions some contract matters, configuration control, a minimum of three design reviews, software development plan, testing details. References use AMC Pamphlet 70-4, "Software Acquisition - A Guide for the Material Developer", MILSTDs 483/490 and Data Item Descriptions.

KNOWN STUDIES NOT INCLUDED IN ABOVE SUMMARY

1. "Findings and Recommendations of the Joint Logistics Commanders' Software Reliability Workshop (SRWG Report)," two volumes. November 1975. NTIS document AD-018881 (2).

2. A short six-week study by Dr. Edwin B. Stear, conducted by him personally just as he was departing his post as Air Force Chief Scientist, and done for General Tom Marsh, Commander, AFSC, at the time. There was no written report but a briefing [dated March 14, 1983] was presented to General Marsh and others.

The study addresses the same general range of topics as other studies, although in some instances, it subdivides them differently or makes slightly different recommendations in regard to remedying perceived shortfalls.

Table A-1 is a summary of the recommendations of the 17 reports described above. The names of the reports, their years of publication, and their numbers (as given above) are listed vertically. The left-hand column lists the recommendations of each report. The letters scattered throughout this table are item designators in the original document and correspond to the letters listed in the preceding appendix.

Source: Organization, or principal name	CARTER	DEFJOUR	HEFFDFN	WALDEN	RAND	GAO	GAO	HULLER	SAWYER	DRUFFEL	SCHAEFER	AF/SAB	ZRAKET	VICK	SELIN	BROOKS	AFDCSLE
Year of document:	'75	'75	'77	'78	'80	'81	'81	'81	'81	'82	'83	'83	'84	'84	'87	'87	'88
RECOMMENDATION & GROUPING	DOCUMENT NUMBER																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
REQUIREMENTS DEFINITION																	
Up front contract for requirements def			A									A					
Better languages for rqments definition														B			
Rapid prototyping to refine rqments													B	D			
DoD institutionalize prototyping																T	
DoD require iterative rqments setting																S	
Each service provide prototyping facility																V	
Include support rqments initially																	B
ACQUISITION	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Update USAF rules and regs re software acq									C								
Tech transfer during acquisition												L					
Incremental delivery			B														
Adapt procurement processes to software				C					B								
Create Centers of Expertise for acq and mgment														E			
Alternate acquisition strategies														F			
Use requirements contracts as useful															F		
Assure quality and training of field acq/purch people															J		
Evolutionary acq to reduce risk																K	

NOTE: Letter is the item designator in the original document (see Appendix A).

TABLE A-1 SUMMARY OF RECOMMENDATIONS OF PREVIOUS SOFTWARE STUDIES

TOOLS and SDE

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Advance software tools			D														
Engineer SDE														A			
Enforce SDE use in all contracts														A			
Use intelligent DBMS														B			
Use KBSA as part of SDE														C			
More money for tools												J					
R&D tools for new architectures														K			
Try to exploit comml tools																M	

DEVELOPMENT PROCESS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Freeze software methodology				D													
Use modern development technology				A													
R&D the software process												B/K					A
R&D test case design														B			
Network the design and mgment team														B			
Collect data re software development				E													
Use V&V				G								M					
Software tech center												F					
DoD develop metrics for QA et al																P	
DoD develop metrics to monitor progress																Q	
DoD kill all vestige of waterfalls																T	
Keep user and supporter involved																	C

PERSONNEL

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Extended assignments to SPO			E														
Train software people			F														
More software people												B					
Career path for people												D				AG	
Better incentives for software/managers														H			
Better training/educ or software/managers														H		AH	
Better selection procedures for assignments														I			
Better trained acq people in MAJCOMs															D		
Have assignments parallel MAJCOM dev effort															E		
Plan to live with the shortage																AD	
Track personnel skills																AF	
Special efforts to acquire, train and retain																	F

POLICY

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
AFSC focal point for comp ters			H														
Mak oftware a high tech risk matter				B													
OMB guidelines for system acquisition						x											
Mission primacy for ADP systems									A								
DCS-level manager												E					
Policy on risk management												G					
Allow for concurrent development															A		
Decentralize some acq to MAJCOMs															B		
Fund via support and development commands															C		
DoD establish 4 classes of acquisition																L	
DoD create policy and and procedure for each																L	
DoD develop contract profit incentives																O	
DoD develop productivity incentives																N	
DoD mandate risk mgment																U	
DoD mandate involvement of users in incremental development																X	

DoD SOFTWARE INITIATIVE

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
DoD software initiative										A							
Plan for initiative										B							
Broad scope of initiative										C							
Formal coord mechanism																A	

MANAGEMENT

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Improve management procedures												C					
Better tools for managers														G			
Risk management procedures and tools												H					
Oversight management												I					
Top level emphasis for visibility															H		
Share lessons learned																	D
Create guides and pamphlets																	E

POST-DEPLOYMENT LCS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LCS during devlopment			C														
Use SDE through LCS														B			
Post-deployment support personnel												O					
More funding for POSS												P					
Early participation by support organization												Q					
Each service provide ability for OTE and LCS evals																W	

INTEGRITY	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Software integrity methodology												N					
Management authority for system integrity												R					
Broader management overview for software matters													A				

REUSABLE SOFTWARE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Reuse software													D				
Reuse tools														N			
Reuse models														N			
Reuse subsystems														N			
Reuse Ada packages														N			
DoD incentivize reuse																Y/Z	
Reward reuse and COTS software																AA	
SEI establish module market																AB	
SEI establish standards for Ada modules																AC	

TECHNOLOGY	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Improve tech base			G														
New standard machine for Ada														L			
R&D for new architectures														M			
Transfer AI technology													C				

STARS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Rehost and improve																B	
New goals																C	
Use STARS goals to improve selected programs																D	

ADA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Commit DoD mgment																H	
DoD increase funding to Ada matters																I	

MISCELLANEOUS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Don't expect software to be sole integration point				F													
Coordinate peace and wartime needs															G		
DoD address data rights																R	
Support JLC efforts re software																	G

APPENDIX B

SCHEDULE OF MEETINGS

First Meeting
Electronic Systems Division
Hanscom AFB, Massachusetts
Building 1704, Room 204
9-10 June 1988

Agenda

Thursday, 9 June 1988

0830	Committee Convenes/Introductions	Dr. Walter Beam, Chairman
0845	Administrative Announcements	Vernon H. Miles, Director
0900	Global Decision Support System (GDSS)	Maj Paul Brusseau
1000	Break	
1015	427M	Mr. Bob Kent
1115	Over the Horizon-Backscatter (OTH-B)	Mr. Ed Bensley
1130	Working Lunch	
1245	PAVE PAWS	Mr. Tom Pucci
1345	Break	
1400	SACDIN	Maj Wayne Balcom
1500	ESD/MITRE Software Center	Dr. Dick Sylvester
1600	Executive Session	Dr. Beam
1900	Cocktails	Hanscom AFB O'Club
1930	Dinner	

Friday, 10 June 1988

0830	Committee Convenes	Dr. Beam

0900	ESD Prototyping/Command Center Evaluation System	Col Jack Ferguson
0930	Software Procurement Issues	Dr. Winston Royce
1030	RADC Software Engineering Program	Mr. Sam DiNitto
1200	Lunch	Hanscom AFB O'Club
1330	Executive Session	Dr. Beam
1500	Adjourn	

Second Meeting
Aeronautical Systems Division
Wright-Patterson AFB, Ohio
Building 14, Room 222
22-24 June 1988

Agenda

Wednesday, 22 June 1988

0830	Committee Convenes/Introductions	Dr. Walter Beam, Chairman
0835	Welcome/Opening Remarks	General Loh
0845	Committee Caucus	Dr. Beam/Mr. Miles
0930	F-16 Software Development	Mr. LeMaster
1000	Discussion	
1015	Break	
1030	ATF Software Development	Mr. Lang
1115	Discussion	
1130	Software Development Integrity Program	Philip Babel
1200	Discussion	
1215	Lunch	
1330	Software Development Capability/ Capacity Review	Mr. Bernard

1400	Executive Seminar on Software Acquistion	Philip Babel
1500	Discussion	
1515	Break	
1530	Executive Committee Session	
1630	Meeting Adjourns	

Thursday, 23 June 1988, Bldg 14, Room 222

0830	Committee Convenes	
0845	B-1 Software Development	Capt. Clark
0930	Discussion	
0945	Software Technology	Mr. Harris
1030	Discussion	
1045	Break	
1100	Artificial Intelligence	Philip Babel
1145	Discussion	
1200	Lunch	
1315	Common ADA Missile Packages and Other Armament Division Software Issues	Ms. Anderson
1415	AFLC Perspective and DoD 2167/2167A	Mr. Kvenvold
1515	Break	
1530	AFLC Presentation Continues	
1630	Executive Committee Session	
1730	Meeting Adjourns	

Friday, 24 June 1988, Bldg 14, Room 203

0830	Formal AFIT Software Training Programs	Dr. Ferens

0930	Executive Committee Session	
1200	Lunch	
1300	Executive Committee Session	
1500	Adjourn	

Third Meeting
Beckman Center/NAS
Irvine CA 92713
11-22 July 1988

Agenda

	Tuesday, 12 July 88	
1300	Ada/STARS/SEI Charters	Terry Courtwright
	Wednesday, 13 July 88	
1300	Electronic Combat Software	Mark Pitts/Phil Babel ICEWMD (ASD/RWA)
1400	SD Perspective	Capt. Taint (SD/ALR)
1500	STARS	Col. Greene
	Thursday, 14 July 88	
1300	AFCC Perspective	Mr. Seay (Tinker AFB)
1400	SPACECOM Perspective	LTC Hassebrock (SC/LKW)
1500	TAC Perspective	Capt. Sowell
1600	SAC Perspective	Capt. Sly (Software Testing)
	Monday, 18 July 88	
am	Prepare Text for Report	
1300	DoD 287 Handbook Discussions	Capt. Romano

Tuesday, 19 July 1988

All day	Prepare Text for Report	
1700	Draft Text Due	

Wednesday, 20 July 1988

am	Prepare Briefing Charts	
1300	Review Draft Briefing	**Presentation by Preparers**

Thursday, 21 July 1988

All day	Review and Develop Draft Report	
1830	Social Hour/Dinner	

Friday, 22 July 1988

0830	Dry Run of Final Briefing	**Presentation by Walter Beam**
	Review Draft Report	
1200		**Adjourn**

Fourth Meeting
National Research Council
2001 Wisconsin Avenue
Green Bldg, Room 118
17-18 August 1988

Agenda

Wednesday, 17 August 1988

0800	Work Session
1200	Lunch
1300	Work Session (continued)
1800	Dinner

Thursday, 18 August 1988

0800 Work Session

1200 Lunch

1300 Work Session (continued)

1700 Adjourn

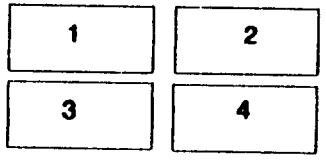
APPENDIX C

BRIEFING CHARTS

Adapting Software Development Policies to Modern Technology

an Air Force Studies Board study for Commander, AF Systems Command. Presentation 19 August 1988 by Dr. W. R. Beam

All charts organized on pages
as shown below



Adapting Software Development Policies
to Modern Technology

1988 SUMMER STUDY: "ADAPTING SOFTWARE DEVELOPMENT POLICIES TO MODERN TECHNOLOGY"

19 Aug 88

BRIEFING OUTLINE

Committee tasking, personnel, meetings
Previous studies
Software perspectives

Focal areas of study (status, findings, and recommendations in)

Acquisition process

Acquisition practices

Software Engineering tools and techniques

In-house software expertise

Software technology program

19 AUG 88-1

TASKS:

1. Study recent software studies & why they did not solve software development problems

2. Assess current and past development programs and identify software deficiencies

3. Investigate methods for software user and developer to work closer...

4. Consider role of prototyping and evolutionary development

5. Evaluate alternative development process models

6. Determine how adequate controls for testability, reliability, survivability can be made part of an incremental flexible development approach.

19 AUG 88-2

Committee :

Walter R. Beam (Chairman) George Mason U.
Robert L. Edge - Consultant
David J. Farber - U. Of Pennsylvania
C Cordell Green - Kestrel Institute
Richard J. Sylvester - MITRE Corp.
Joseph E. Urban - U. of Miami
Willis M. Ware - Rand Corp

Liaison Members:

Barry Boehm - TRW Corp.
Philip S. Babel - ASD/EN
Sam DeNitto - RADC/COE
Lt/C T. Courtwright - AFSC/XTKC
Winston W. Royce - Lockheed Aero. Syste.
Mary M. Shaw - Software Engrg Inst.
Study Director: Vernon H. Miles - NRC

19 AUG 88-3

ORGANIZATIONS VISITED OR HEARD FROM:

VISITS:

-Electronic Systems Division (June 8-10)

-Aeronautical Systems Division & AD, AFLC, (June 22-24)

-Warner Robins ALC (July 6)

- AFSC (1 Aug, AFECO; 8 Aug, AFSC/PLR)

PRESENTATIONS AT IRVINE DURING SUMMER STUDY PERIOD (July 11-22)

- Space Division
- SAC
- TAC
- SPACECOM
- AFCC
- STARS
- U. Cal. Irvine (Software Safety)

PREVIOUS STUDIES

-Early 1970s to 1988
-At request of JLC,USAF, OSD, GAO
-Substantially consistent
- Reflected some emerging ideas & trends
  - Reusability
  - Software integrity
  - Development tools & environments
-Reflected escalating life-cycle support concerns in USAF

Six issues repeatedly addressed:
- up-front requirement definition
- acquisition procedures
- acquisition policy
- development process
- personnel
- project oversight management

Seen as mostly non-technical issues, but with technical aspects.

Failed to "solve software problem"
- growth of the problems not anticipated
- actions not sufficiently aggressive

and the problems both grow and change.

AFSC software perspectives -1988

- Software now dominates system functionality (did not, a decade ago)
- Good/poor software will now make/break system
- Systems thinking still dominated by hardware
- Many key system players weak in software (and the converse)
- Software a handy whipping boy for other woes
- Software often passed difficult hardware problems to "fix"; even if fixable, leads to overruns
- Software not properly recognized as a long-lead item, requirements late, software started late

Special concerns: Ada, STARS, SEI

- Ada is good news (really improves software process), bad news (compilers may not be available, real-time performance limiting)

- STARS results should benefit USAF, though indirectly via contractors- and long term

- Treat SEI as another reservoir of expertise which AFSC should try to tap (assessment of contractors, definition of SEE capabilities)

(Uncertainty: USAF access to SEI and STARS under DARPA management.)

19 AUG 88-8

ACQUISITION

PROCESS

ACQUISITION PROCESS

Sequential process model (waterfall) implied by MIL-STD-2167 et al

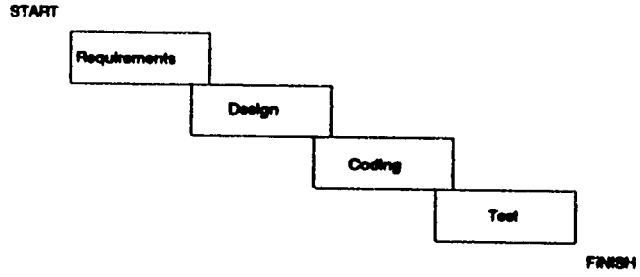
- Assumes doable, fixed requirements from start
- Open loop, driven by unrealistic specs and schedules and by documentation, not by demonstrable results

Widely attempted - often fails

- Contractors too often follow slavishly what they believe to be desires of agency

19 AUG 88-9

WATERFALL MODEL OF DEVELOPMENT



19 AUG 88-9A

FINDINGS

-Waterfall model works for precedented systems
-Has high risk for unprecedented systems
  Does not emphasize systemsoftware risk reduction:
  -In requirements definition
  -In requirements-design evolution
  -In creation of effective system design teams

Alternative acquisition and development process models are necessary for unprecedented systems

19 AUG 88-10

CHARACTERISTICS OF A PRECEDENTED SYSTEM

(must have all three)

1. Stable system and software requirements, not significantly different from those of previously developed system(s)

2. Digital system architecture and software design mature and known able to meet reqts

3. Systems engineering and software devt teams must have previous experience with similar systems and must communicate effectively.

19 AUG 88-10A

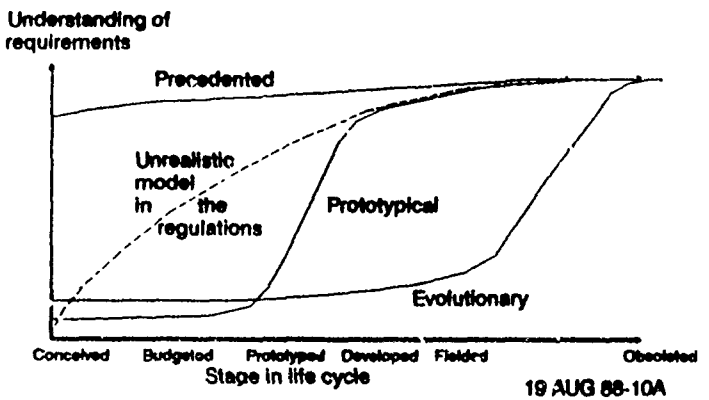
RECOMMENDATIONS:

Address risk reduction through alternative processes
- DEM/VAL phase to validate reqts, prove design, build experience
- two-phase competitive procurement to reduce performance, cost, schedule risks
- parallel technology programs to maintain warm industry base for key software applications, prove requirements, and demonstrate system realizability in areas where reqts evolve quickly

Describe alternative models in AFRs 800-2 & 800-14, and HBK-287

19 AUG 88-11

EVOLUTION OF SYSTEM REQ'TS



19 AUG 88-10A

WHAT CONTRACTORS SHOULD DEMONSTRATE BEFORE FSD

1. Architectural proof-of-concept
- digital hardware/software design (prototyped)
- connection to requirements traced
- timing, storage, margins based on meas'ts

2. Software methodology proof-of -compliance
- prior usage, software engineering exercise
- new methods demonstrated
- methodology enforced, e.g. thru a software engineering environment

3. System and software skills readiness
- demonstrated experience/training in applic'n
- proven experience/training in methods/tools

19 AUG 88-11B

ACQUISITION

PRACTICES

ACQUISITION PRACTICES

Currently imposed process criticized by contractors
- waterfall model
-MIL-STD-1521,2167A
- other specs, standards blindly imposed
- contractors afraid to rock the boat

Firm fixed price contracts impose on unprecedented system developments

SPOs suggest "more" user involvement
- little insight that user involvement also should be tailored

Motivation NOT to tailor aquisition
- personal risk without reward potential
- lacking knowhow, not comfortable

19 AUG 88-12

FINDINGS

- Acquisitions often untailored to system
- Poorly structured user involvement
- User or maintainer seldom considered for IV&V role (as directed in an AFR Supplement)
- User/maintainer often plays little or no role in life-cycle support plans
- Firm Fixed Price contract type too often selected for acquiring unprecedented systems
- Software advanced development may be done during FSD

19 AUG 88-13

RECOMMENDATIONS

Motivate and train for acquisition tailoring
- Issue policy statement, coriductAFSC workshops
- Expedite handbooks for 2167A and 2168

Tailor user and maintainer involvement to system

Fit contract type to state of unknowns
- No FFP for unprecedented systems...
  resolve unknowns first thru DEM/VAL or equiv.

Treat software, hardware and system unknowns unequivocally in determining readiness for FSD

19 AUG 88-14

STRAWMAN PROCESS-MODEL SELECTION FACTORS

Examples	Understanding of requirements	Applicable commercial software	System architecture understood?	Application criticality & risk	Future evolution understood?	Candidate process model
Inventory control resource monitor	Well under stood	Complete application packages	Yes	Low	Yes	Acquire COTS
Updating of fielded system	Well under-stood	Probably none	Well understood	Low to moderate	Reasonably well	Waterfall (with Ada)
MIS portion of command & control system	Still converging	DBMS, proto typing tools, screen gen.	Usually not	Low to moderate	Generally but not in detail	Evolutionary development (build, rebuild)
Expert system, intel analysis tool	Still converging	AI shells	Yes, but with limited coupling	Moderate	Not at all	Exploratory, for limited field use.
Most C-cubed, avionics	Still converging	Prototyp ing tools,VHLLs (for evolution)	In part	Moderate to high	Generally but not in detail	Prototype, followed by waterfall
Large distributed real time display oriented systems	Only near-term under-stood	No	No	Moderate to high	Poorly	Incremental development

19 AUG 88-14A

SOFTWARE ENGINEERING TOOLS AND TECHNIQUES

SOFTWARE ENGINEERING TOOLS AND TECHNIQUES

(Should be integrated as Software Engineering Environments.)

Need automated tools for:
- rapid, interactive software design exploration
- inter-group communication
- continuity of information through life cycle
- design evolution and standardization
- test definition, support & reporting
- improving the contracting process:
  - Progress auditing
  - Metrics collection
  - Resource planning

19 AUG 88-15

S.E.E. TOOLSETS - DEFINITION

Minimum (kernel) - configuration management, requirements traceability, data base support, access to commercial packages, documentation support, programmer's notebook, compiler/debugger/etc. platform, test vehicle, controlled remote access

Design exploration set - design analysis tools libraries of reusable code and simulators, very high level language support, syntax directed editor, application oriented language(s), support for executable specification, test effectiveness tools, quality assessment and correctness tools.

Management support set - acquisition process models, schedule analysis, action and error tracking, coupling to SOW, WBS, priorities, deliverables, personnel, metrics collector, early warning, acquisition management communications.

19 AUG 88-15A

FINDINGS

- SEEs available in a few varieties, though none complete
- Widely believed cost-effective for software development (at $30K/person)
- Major contractors & NASA (Space Station) now acquiring and installing SEEs (120 approved tools for NASA system)
- JAIWG (ICNIA, INEWS, etc.)establishing GFE'd SEE for its contractors.
- SEE capabilities essential when dealing with large, distributed developments
- Provide tools which can help AFSC
- Time is right for AFSC to get more involved

19 AUG 88-16

RECOMMENDATIONS

- AFSC require contractors to use appropriate SEEs (certain minimum capabilities) in all acquisitions with major software devt (e.g. for > 1 dozen people)
- Developers of unprecedented systems required to have and use design exploration tools, in addition
- AFSC acquire and use a SEE which supports computer-based documentation and management toolset
- Contractors to deliver timely software engineering data in machine readable formats compatible with AFSC systems

19 AUG 88- 17

IN-HOUSE

SOFTWARE

EXPERTISE

FINDINGS - IN-HOUSE EXPERTISE

- AFSC has about 320 identifiable software specialty officers, unknown no. of civilians or systems engineers with good software expertise
- Few senior people, in an already skinny field
- Some experienced people hiding in other classifications with better promotability
- No doubt that need for in-house AFSC software expertise on the rise, but no Air Force or AFSC program to satisfy need *quickly*

19 AUG 88-18

RECOMMENDATIONS
(including both military and civilians)

- Organize and use Ace Advisory Team(s)
  - Identify "Aces" (knowledge, experience)
  - Organization and assign centrally
  - Use at key stages in programs (pre-RFP, proposal evaluation, critical stages during acquisition.
  - Help programs start and stay healthy
- Identify/develop System Engrs with strong software skills, Software Engineers with system competence.
- Intensively manage this scarce resource
  - Through Advisory Team(s) activities
  - Tailored career mgt
  - Special monitoring

19 AUG 88-19

SOFTWARE

TECHNOLOGY

FINDINGS - SOFTWARE TECHNOLOGY

- Commercially available software and research results can improve the software development processes
- Developers and buyers reluctant to adopt these for various reasons (NIH, personnel limitations, political issues).
- SEEs are good media for transfer of software technology (in form of tools)
- Software research, development, production work underfunded by factor of 2-4, by comparison with other key technology areas.

19 AUG 88-20

Software Technology Transfer (exemplary)

TECHNOLOGY	BENEFIT	APPROACH
PROGRAMMING & SPECIFICATION LANGUAGE	Reduce comm gap by moving language(s) closer to user contexts	Graphics languages HLLs, VHLLs, applicn specific langs, formal specifications
CONSISTENCY CONTROL	Reduce inconsistencies and related errors, improve requirements traceability	Global (SEE) typing and definition controls, consistency checkers, translation to application terminology
PROCESS MODEL SUPPORT	Timely tracking and monitoring, improved communication and management, adaptation to unprecedented situations	Formal process model rule definition and conversion to SEE algorithms
EARLY PROTOTYPING AND DEMONSTRATION	Quicker convergence of requirements	Prototyping tools for display systems, signal processing, information fusion

19 AUG 88-20A

RECOMMENDATIONS

- Use SEE as vehicle for technology transition, (for process-oriented technologies, e.g. design exploration tools)
- Continue to fund language evolution, moving programming language closer to user viewpoint
- Support evolution of management-support tools as part of software efforts
- Initiate a MANTECH-like Program Element in USAF to enable ikmproved productivity in software development
- Encourage commercialization of software process technologies
- Spend as much in software technology base as it can sensibly absorb, in next 5 years

19 AUG 88-21

CONCLUSION

PROCESS - apply the right process to each aquisition, to minimize risks

PEOPLE - build software/system capabilities needed for a software-dominated system environment

TECHNOLOGY - recognize the nature of software technologies and exploit them for better systems, on cost, on schedule

19 AUG 88-22

GLOSSARY

Ada The DoD standard higher-level programming language, developed starting in 1977.

greybeard team Senior experts.

non-iterative process A process that does not repeatedly carry out some part in an attempt to improve or refine it.

red team An in-house group that "scrubs" the program. It checks for weaknesses and reports to management.

robust system A system that gracefully handles changes in itself and its environment; opposite of "brittle."

unprecedented system A system for which there has been no precedent in the form of similar systems or systems performing the same functions, or for which the design teams lack full or applicable system design experience.

waterfall model A process for system or software development in which the entire project is separated into phases and the work of each phase completed before the next one is begun. In particular, this process is non-iterative except within phases.

ACRONYMS

4GL	Fourth Generation (programming) Language
ADP	automatic data processing
AFCC	Air Force Communications Command
AFLC	Air Force Logistics Command
AFOTEC	Air Force Operational Test and Evaluation Center
AFR	Air Force Regulation
AFSC	Air Force Systems Command
AFSPACECOM	Air Force Space Command
AI	artificial intelligence
AIRMICS	Army Institute for Research in Management, Information, Communications, and Computer Science
AJPO	Ada Joint Program Office
ALC	Air Logistics Command
APL	A Programming Language
ASD	Aeronautical Systems Division
ATC	Air Training Command
C^3I	command, control, communications, and intelligence
CINC	Commander-in-Chief
CPCI	computer program configuration item
CRWG	computer resources working group

CSI	critical software issue
DARPA	Defense Advanced Research Projects Agency
DCS	Defense Communications System
	Deputy Chief of Staff
DEM/VAL	demonstration/validation
DoD	Department of Defense
DoD-STD	Department of Defense Standard
DMA	Defense Mapping Agency
DSB	Defense Science Board
DSMC	Defense Systems Management College
DT&E	development, test, and evaluation
ESD	Electronic Systems Division
EW	electronic warfare
FCRC	federal contract research center
FSD	full scale development
FSED	full scale engineering development
GAO	General Accounting Office
GFE	government furnished equipment
HLL	high level language
IR&D	industrial research and development
IV&V	independent validation and verification
JAIWG	joint avionics integration working group
JLC	Joint Logistics Commanders
KBSA	knowledge-based software assistant
LCS	life cycle support
MAJCOM	major command
MANTECH	manufacturing technology
MCC	mission control complex
MIL-STD	Military Standard
MSD	Munitions Systems Division
OMB	Office of Management and Budget
P^3I	pre-planned product improvement
PDR	preliminary design review
RADC	Rome Air Development Center
SAB	Air Force Scientific Advisory Board
SAC	Strategic Air Command
SACDIN	SAC Digital Information Network
SCP	strategic computing program
SECCEP	Scientist and Engineer Career Enhancement Program
SEE	software engineering environment
SEI	Software Engineering Institute
SETL	Set (theoretical programming) language
SPC	Software Productivity Consortium
SPO	system program office
SSD	Space Systems Division
SSE	software support environment
STARS	software technology for available, reliable systems
TAC	Tactical Air Command
VHLL	very high level (programming) language
WIS	worldwide information system